Bull

AIX 4.3 Guide d’optimisation

AIX

REFERENCE
86 F2 72AP 04

Bull

Bull

AIX 4.3 Guide d’optimisation

AIX

Logiciel

Octobre 1999

BULL ELECTRONICS ANGERS
CEDOC
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

REFERENCE
86 F2 72AP 04

Imprimé en France

Vos suggestions sur la forme et le fond de ce manuel seront les bienvenues. Une feuille destinée à recevoir vos remarques se trouve à la fin de ce document.

Pour commander d'autres exemplaires de ce manuel ou d'autres publications techniques Bull, veuillez utiliser le bon de commande également fourni en fin de manuel.

Marques déposées

Toutes les marques déposées sont la propriété de leurs titulaires respectifs.

AIX® est une marque déposée d'IBM Corp. et est utilisée sous licence.

UNIX est une marque déposée licenciée exclusivement par X/Open Company Ltd

An 2000

Le produit documenté dans ce manuel est prêt pour l'An 2000.

Table des matières

A propos de ce manuel

Avertissement: Dans ce manuel, ce qui s'applique au DPX/20 s'applique également aux systèmes Unix suivants : ESCALA et ESTRELLA.

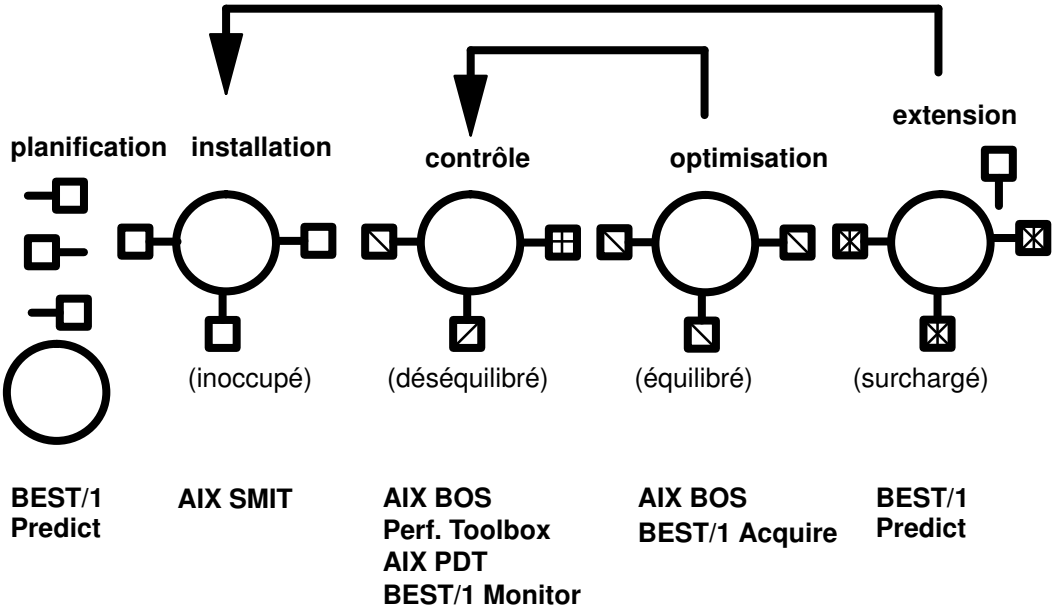
Ce manuel, *AIX - Guide d'optimisation*, donne des informations sur les concepts, les outils et les techniques d'évaluation et d'optimisation d'AIX sur les systèmes ESCALA. Il traite de la conception et de l'implantation du système et des applications, ainsi que de l'optimisation post-implantation de l'utilisation de la CPU et de la mémoire, et des E/S disque et de communication. La plupart des conseils d'optimisation ont été testés ou validés sur la version 3.2.5. Des précisions sur AIX version 4.1 sont également fournies. Les informations qui ne s'appliquent qu'à AIX version 4.1 sont signalées dans le texte.

Destiné aux administrateurs système, programmeurs et utilisateurs finals concernés par l'optimisation des systèmes AIX. Ce manuel suppose une bonne connaissance de l'environnement d'exploitation AIX. Des sections de présentation permettent aux utilisateurs moins chevronnés d'acquérir les notions indispensables et aux autres de se familiariser avec la terminologie propre à l'optimisation.

Ce manuel est une version totalement revue du manuel *AIX version 3.2 Performance Monitoring and Tuning Guide*, tant au niveau de la structure que des conseils fournis. Notons notamment les modifications intervenues au niveau du conditionnement et du contenu des outils d'analyse de performances dans AIX version 4.1.

Gestion des performances AIX : structure

Il existe des outils pour chaque phase de la gestion de l'optimisation d'AIX. Certains sont fournis par Bull, d'autres par des fournisseurs tiers. La figure suivante illustre les phases de cette analyse dans un environnement réseau local simple et indique les outils applicables à chaque phase. Ce manuel donne également des informations sur chaque phase.



Phases d'analyse des performances et outils correspondants

Outils et documentation

Le conditionnement des outils de performance permet à l'analyste de n'installer sur un système que les outils requis pour contrôler et optimiser ce système. Les modules et leurs principales fonctions sont décrits ci-après.

BEST/1

BEST/1 est un outil de planification de la capacité qui exploite des modèles de mise en file d'attente pour préévaluer les performances d'une configuration donnée lors du traitement d'une charge donnée. Les prévisions peuvent être fondées sur :

- les descriptions de charge issues d'une application,
- les données de la charge collectées par surveillance des systèmes existants.

BEST/1 est constitué de trois modules principaux :

Collect	Collecte des informations détaillées sur le traitement d'une charge par un système existant.
Analyze	Transforme les informations détaillées en comptes rendus et en un modèle de mise en file d'attente de l'activité de traitement de la charge.
Predict	Utilise le modèle de mise en file d'attente pour estimer les effets de modification de la charge ou de la configuration sur les performances.

BEST/1 pour UNIX est un produit de BGS Systems, Inc. Vous pouvez contacter BGS Systems au 1–800–891–0000 (aux Etats–Unis).

Outil de diagnostic des performances AIX (PDT)

L'outil Performance Diagnostic Tool (PDT), élément installable en option d'AIX version 4.1, évalue la configuration du système et détermine les principaux axes d'utilisation des ressources. Si PDT détecte un problème de performance, réel ou potentiel, il l'indique à l'administrateur système. Ce manuel traite des fonctions de PDT.

Système d'exploitation de base AIX (BOS)

Le système d'exploitation de base AIX propose un certain nombre d'outils de contrôle et d'optimisation, historiquement intégrés aux systèmes UNIX ou requis pour gérer les caractéristiques d'implantation propres à AIX. Voici les fonctions et commandes BOS essentielles à l'analyse des performances :

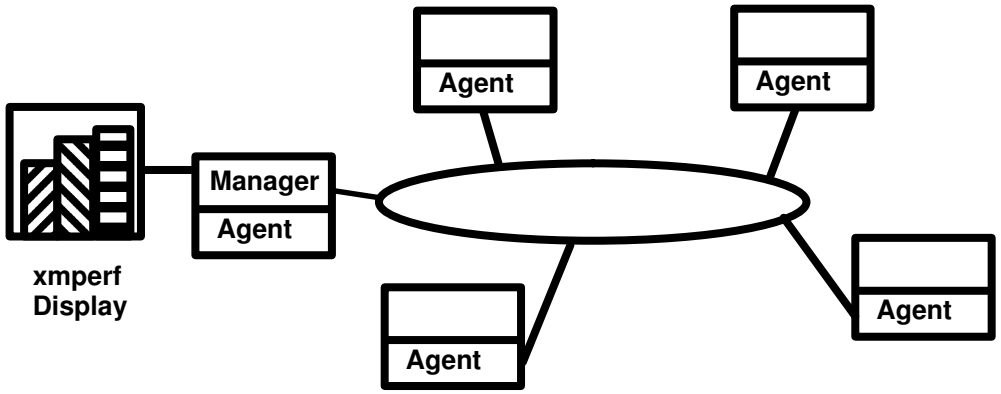
iostat	Rapporte les statistiques relatives à la CPU et aux E/S.
lsattr	Affiche les attributs des unités.
lslv	Affiche des informations sur un volume logique ou les affectations de volumes logiques sur un volume physique.
netstat	Affiche le contenu des structures de données relatives au réseau.
nfsstat	Affiche des statistiques sur les activités NFS (Network File System) et RPC (Remote Procedure Call).
nice	Exécute une commande à un niveau de priorité supérieur ou inférieur à son niveau normal.
no	Affiche ou définit des options réseau.
ps	Affiche l'état des processus.
renice	Modifie la priorité d'un ou de plusieurs processus.
reorgvg	Réorganise l'affectation des partitions physiques dans un groupe de volumes.
sar	Collecte et rend compte ou enregistre des informations sur l'activité du système.

time	Imprime la durée d'exécution écoulée et les temps de traitement utilisateur et système attribués à une commande.
trace	Enregistre et rend compte d'événements système sélectionnés.
vmstat	Rend compte de l'activité de la mémoire virtuelle et d'autres statistiques système.

Les commandes BOS AIX sont décrites en détail dans les manuels de référence de commandes d'AIX.

Boîte à outils Performance Toolbox (PTX)

La boîte à outils Performance Toolbox for AIX (PTX) propose des outils de contrôle et d'optimisation de l'activité de systèmes locaux et distants. Ce produit sous licence est constitué de deux composants principaux : PTX Manager et PTX Agent. PTX Agent est disponible sous licence distincte sous l'appellation Performance Aide for AIX. La figure illustre une configuration de réseau local simplifié où PTX Manager contrôle l'activité de plusieurs systèmes.



Configuration de réseau local exploitant Performance Toolbox

L'objectif principal de PTX Manager est de collecter et d'afficher les données issues de différents systèmes de la configuration. **xmperf** est le principal programme dédié à cet objectif. Le principal programme utilisé par l'Agent pour collecter et transmettre les données au Manager est **xmservd**.

Sous AIX version 4.1, outre les composants de base PTX, les produits sous licence Performance Toolbox for AIX et Performance Aide for AIX proposent un ensemble d'outils de contrôle et d'optimisation distinct, dont l'essentiel fait partie de BOS version 3.2.5 :

fdpr	Optimise un programme exécutable pour une charge de travail donnée.
filemon	Exploite l'utilitaire de suivi pour contrôler et rendre compte de l'activité du système de fichiers AIX.
fileplace	Affiche la position des blocs d'un fichier dans des volumes physiques ou logiques.
lockstat	Affiche des statistiques relatives aux verrouillages de noyau.
netpmon	Exploite l'utilitaire de suivi pour rendre compte de l'utilisation de CPU par les E/S réseau et les activités relatives au réseau.
rmss	Simule des systèmes dotés de mémoires de tailles différentes pour tester les performances.
svmon	Capture et analyse les informations concernant l'exploitation de la mémoire virtuelle.
syscalls	Enregistre et compte les appels système.

tprof	Exploite l'utilitaire de suivi pour rendre compte de l'utilisation de CPU au niveau module et instructions du code source.
bf	Rend compte des schémas d'accès mémoire des processus (AIX version 4.1 seulement).
stem	Autorise l'instrumentation des entrées/sorties au niveau sous-routine des exécutables existants (AIX version 4.1 seulement).

La principale documentation relative aux commandes et fonctions de PTX est le manuel *Performance Toolbox 1.2 and 2.1 for AIX: User's Guide*, bien que la syntaxe des outils cités plus haut soit documentée dans le manuel *AIX Commands Reference*. L'utilisation des commandes de la liste est intégrée aux divers scénarios de diagnostics et d'optimisation de ce manuel.

Outil de collecte des données de performance PMR AIX (PerfPMR)

L'outil de collecte des données de performance PMR AIX (PerfPMR) sert à collecter les données de performances et de configuration, à joindre à un rapport signalant un défaut supposé au niveau des performances AIX. Ce manuel contient une description des fonctions de PerfPMR.

Outils non pris en charge sous AIX version 4.1

Le programme **rmap** de compte rendu et de réduction des données de suivi n'est pas pris en charge par AIX version 4.1.

Mode d'emploi du manuel

Contenu

Ce manuel est composé des chapitres et annexes suivants :

- Chapitre 1 : Performances : introduction à l'analyse des performances. Pour les utilisateurs déjà familiers des techniques d'optimisation, ce chapitre a surtout l'intérêt de présenter la terminologie propre à ce type d'opération.
- Chapitre 2 : Gestion des ressources AIX : structures et algorithmes de base des principaux composants de gestion des ressources AIX.
- Chapitre 3 : Introduction au multitraitement : présentation des performances des systèmes multi-traitement.
- Chapitre 4 : Planification, conception et implantation : éléments relatifs aux performances à prendre en compte lors de la préparation d'une application.
- Chapitre 5 : Contrôle système et diagnostics des performances : étapes préliminaires de détection d'un problème de performance.
- Chapitre 6 : Contrôle et optimisation de la CPU : techniques de vérification de l'utilisation optimale des ressources CPU.
- Chapitre 7 : Contrôle et optimisation de la mémoire : détermination de la quantité de mémoire réelle et virtuelle effectivement utilisée, et techniques pour éviter ou détecter les insuffisances courantes.
- Chapitre 8 : Contrôle et optimisation des E/S disque : description de la dynamique des E/S disque sous AIX et effet des choix utilisateur sur cette dynamique.
- Chapitre 9 : Contrôle et optimisation des E/S de communication : techniques d'optimisation des différents types d'E/S de communication.
- Chapitre 10 : Optimisation des performances DFS : description des paramètres DFS ayant une incidence sur les performances.
- Chapitre 11 : Analyse des performances avec l'utilitaire de suivi : explication détaillée de l'exploitation de l'utilitaire de suivi, puissant outil d'optimisation et base de nombreux autres outils décrits dans le manuel.

- Chapitre 12 : Outil de diagnostic PDT : description d'un nouvel outil d'AIX version 4.1, qui évalue les configurations au niveau de leur équilibre et tient à jour un historique des données de performance pour en identifier les tendances.
- Chapitre 13 : Gestion d'un possible défaut de performance AIX : description de la marche à suivre pour signaler un possible défaut de performance au niveau d'AIX et du mode de collecte et de transmission des données requises.
- Annexe A : Commandes de contrôle et d'optimisation des performances AIX : liste des commandes AIX les plus utiles pour le contrôle et l'optimisation d'AIX et détail de la syntaxe et des fonctions des commandes **schedtune**, **vmtune**, **pdt_config** et **pdt_report**.
- Annexe B : Sous-routines liées aux performances : description de plusieurs sous-routines liées aux performances.
- Annexe C : Mémoire cache et adressage : aperçu de l'impact des caches sur les performances des programmes.
- Annexe D : Commande ld : techniques d'utilisation de l'éditeur de liens AIX.
- Annexe E : Outils de performance : consommation de ressources et temps de réponse des outils de performance.
- Annexe F : Gestion de la mémoire des applications : distinction entre les versions d'origine et courante des sous-routines **malloc** et **realloc**.
- Annexe G : Bibliothèques partagées : avantages et inconvénients des bibliothèques partagées au niveau des performances.
- Annexe H : Accès à l'horloge du processeur : méthodes d'utilisation de l'horloge du processeur pour calculer des durées.
- Annexe I : Support NLS (National Language Support) : effet de l'utilisation du support NLS (National Language Support) d'AIX sur les performances.
- Annexe J : Récapitulatif des paramètres AIX optimisables : liste des paramètres d'AIX modifiables par l'utilisateur, ayant une incidence directe ou indirecte sur les performances.

Conventions typographiques

Voici les conventions typographiques adoptées dans ce manuel :

Gras	Commandes, sous-routines, mots-clés, fichiers, répertoires et autres éléments dont le nom est prédéfini par le système. Objets graphiques à sélectionner (boutons, étiquettes, icônes, etc.).
Italique	Paramètres dont le nom ou la valeur est fourni par l'utilisateur.
`Espacement fixe`	Exemples (de valeurs spécifiques, de texte affiché, de code programme), messages système ou données entrées par l'utilisateur.

ISO 9000

Ce produit répond aux normes qualité ISO 9000.

Bibliographie

Les manuels suivants concernent le contrôle des performances :

- *AIX - Bibliographie*, référence 86 F2 71WE.
- *AIX 4.3 Guide de l'utilisateur : système d'exploitation et unités*, référence 86 F2 97HX.
- *AIX 4.3 Guide d'administration : système d'exploitation et unités*, référence 86 F2 99HX.

- *AIX 4.3 Guide d'administration : communications et réseaux*, référence 86 F2 31JX.
- *AIX Commands Reference*, référence 86 A2 38JX à 86 A2 43JX.
- *AIX General Programming Concepts : Writing and Debugging Programs*, référence 86 A2 34JX.
- *AIX Technical Reference, Volume 1: Base Operating System and Extensions*, référence 86 A2 81AP.
- *AIX Technical Reference, Volume 2: Base Operating System and Extensions*, référence 86 A2 82AP.
- *Performance Toolbox 1.2 and 2.1 for AIX: User's Guide*, référence 86 A2 10AQ.
- *Optimization and Tuning Guide for XL Fortran, XL C and XL C++*, référence 86 A2 99WG
- Comer, D., *Internetworking with TCP/IP Vol I*, 2nd ed., Englewood Cliffs: Prentice-Hall, 1991.
- Ferrari, D., Serazzi, G., and Zeigner, A., *Measurement and Tuning of Computer Systems*, New York: Prentice-Hall, 1983.
- Lazowska, D., Zahorjan, J., Graham, G., and Sevchik, K., *Quantitative System Performance*, New York: Prentice-Hall, 1984.
- Leffler, McKusick, Karels, and Quarterman, *The Design and Implementation of the 4.3 BSD Unix Operating System*, Reading: Addison-Wesley, 1989.
- Smith, C. U., *Performance Engineering of Software Systems*, Reading: Addison-Wesley, 1990.
- Stern, H., *Managing NFS and NIS*, Sebastopol, CA: O'Reilly & Associates, Inc., 1992.

Commande de manuels

Vous pouvez commander ces publications auprès de votre représentant commercial.

Si vous disposez du manuel *AIX - Bibliographie*, consultez–le.

Chapitre 1. Performances

Toute personne exploitant un ordinateur a son opinion sur ses performances. Malheureusement, ces opinions sont souvent basées sur des idées simplistes quant à la dynamique de l'exécution des programmes. Des intuitions non étayées peuvent conduire à des interprétations erronées et coûteuses, sur les capacités d'un système et des solutions aux problèmes de performance perçus.

Ce chapitre décrit le fonctionnement dynamique des programmes et les concepts utiles à l'évaluation des performances d'un système. Elle comporte les sections suivantes :

- Vitesse d'exécution : introduction à la mesure des performances.
- Estimation de la charge : détermination des objectifs de performances.
- Dynamique d'exécution des programmes : identification des points de ralentissement durant l'exécution des programmes.
- Dynamique système : description de l'interaction des programmes.

Vitesse d'exécution

Les termes de "vitesse" et de "rapidité", que l'on continue d'employer pour décrire les ordinateurs, font référence à une réalité aujourd'hui dépassée. Le temps n'est plus en effet où il était possible de lire un programme, de calculer la somme des durées d'exécution de ses instructions pour évaluer son temps d'exécution. Le travail de milliers de programmeurs et d'ingénieurs depuis trente ans a rendu ce type de calcul sinon impossible, du moins dénué de sens.

Les machines d'aujourd'hui sont plus puissantes que leurs ancêtres. Le remplacement des lampes à vide par des circuits intégrés et le raccourcissement significatif des cycles d'exécution ont certes contribué à cette évolution, mais les innombrables nouveautés en matière d'architecture matérielle et logicielle ont été décisives. Chaque avancée en terme de densité de circuit intégré fait progresser d'autant les performances, non seulement parce qu'une même opération est alors exécutable sur une surface moindre à une vitesse d'horloge supérieure, mais aussi parce que le gain d'espace induit offre aux ingénieurs un nouvel espace d'expérimentation. Bref, les machines ont vu leur capacité s'accroître en gagnant à la fois en complexité et en rapidité.

La complexité des machines modernes et des systèmes d'exploitation va de pair avec celle de leur environnement. Outre l'exécution de programmes individuels, les ordinateurs ont aujourd'hui à gérer un nombre variable d'interruptions imprévisibles issues des unités d'E/S et de communication. Au point que les meilleures idées des ingénieurs, basées sur l'hypothèse d'un seul programme exécuté sur une machine autonome, sont aujourd'hui largement rattrapées et dépassées par les impondérables du monde réel. Dès lors qu'il est tenu compte de ce caractère aléatoire, certaines erreurs peuvent être compensées. Les gains et les pertes varient ainsi d'un programme à l'autre au cours du temps.

Le solde net entre les gains et les pertes au niveau logiciel et matériel donne la performance du système. La "rapidité" d'un système est le débit avec lequel il traite une séquence spécifique de demandes. Si les demandes s'imbriquent bien aux architectures matérielle et logicielle du système, on peut dire : "Le système exécute rapidement cette charge de travail." Nous ne pouvons pas dire "Le système *est* rapide"—ou du moins, nous ne le devrions pas.

Estimation de la charge

L'évaluation complète et exacte de la charge d'un système est déterminante pour la prévision et l'analyse de ses performances. Toute variation de la charge a en effet davantage d'incidence sur la mesure des performances qu'une différence au niveau de la vitesse d'horloge CPU ou de la taille de la RAM. La charge du système doit tenir compte non seulement du type et du débit des demandes soumises, mais aussi de l'ensemble des modules logiciels et des programmes d'application internes impliqués.

Il convient de suivre au mieux l'activité des utilisateurs des applications existantes pour disposer d'une évaluation exacte et dynamique de leur interaction avec les stations de travail et les terminaux.

Veillez à inclure dans votre évaluation l'activité "sous–jacente" du système. Par exemple, si le système est doté de systèmes de fichiers NFS fréquemment sollicités par d'autres systèmes, vous devez considérer ces accès comme une portion significative de la charge globale, même si le système ne joue pas officiellement le rôle de "serveur".

Un raccourci risqué : les bancs d'essai standard

Un banc d'essai est une charge normalisée destinée à servir de référence dans la comparaison de systèmes dissemblables. Tout banc d'essai suffisamment éprouvé, devenu "standard de facto de l'industrie", est bien entendu étudié de façon exhaustive par les développeurs : systèmes d'exploitation, compilateurs et, dans certains cas, matériels ont été optimisés pour exploiter ce banc d'essai dans des temps record.

Mais, dans la réalité, les charges de travail concordent rarement avec les algorithmes et l'environnement d'un banc d'essai. En effet, ces bancs d'essai standard, issus d'applications réelles, ont généralement été simplifiés et homogénéisés pour s'adapter à une grande variété de plates–formes matérielles. Par ailleurs, leur environnement d'exploitation a été restreint afin d'offrir des conditions de mesure reproductibles.

Autrement dit, un raisonnement du type "le système A offre un débit en méga–x de 50 % supérieur au système B, donc le système A doit exécuter mon programme 50 % plus vite que le système B" est tentant mais faux. Aucun banc d'essai n'offre une telle applicabilité universelle. La seule utilité réelle d'un banc d'essai est de réduire le champ des systèmes candidats à une évaluation sérieuse. Il n'existe pas de substitut pour réaliser une analyse claire de *votre* charge de travail et des performances systèmes impliquées.

Objectifs de performances

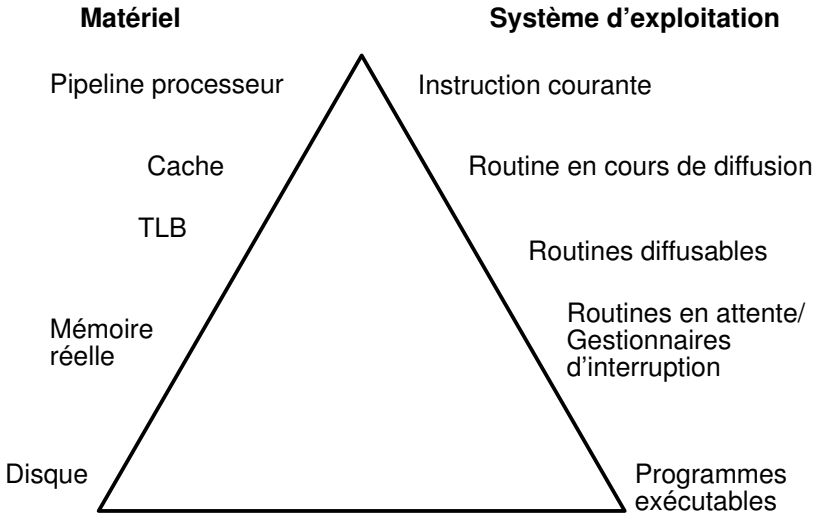
Une fois la charge de travail évaluée, vous pouvez définir vos critères de performances et fixer vos objectifs en conséquence. Les principaux critères de performances globales des systèmes informatiques sont le *temps de réponse* et le *débit*. Le temps de réponse est le laps de temps écoulé entre le lancement d'une opération et l'obtention des informations utiles pour la reprise du travail, alors que le débit est le *nombre* d'opérations comprises dans la charge de travail et pouvant être accomplies par unité de temps. La relation entre ces mesures est complexe. Dans certains cas, vous devez modifier les deux mesures. Dans d'autres, une seule modification suffit à améliorer la situation.

Pour une optimisation planifiée, vous devez déterminer clairement vos objectifs en termes de temps de réponse et de débit pour une charge de travail précise. Faute de quoi, vous risquez de perdre temps et argent pour l'amélioration d'un aspect secondaire du système.

Dynamique d'exécution des programmes

Généralement, un programmeur conçoit son programme comme une séquence continue d'instructions exécutant une fonction spécifique. Beaucoup d'efforts et de créativité ont été déployés, au niveau du système d'exploitation et du matériel, pour conserver aux programmeurs cette vision idéalisée, leur évitant trop de questions sur les notions d'espace, de vitesse et de multiprogrammation/multitraitement. Les programmes reposant sur cette vision illusoire risquent d'être inutilement coûteux (en temps) et inefficaces en termes de performances.

Une analyse claire des performances liées à une charge de travail doit s'appuyer sur un modèle dynamique et non statique de l'exécution d'un programme, comme illustré à la figure Hiérarchie d'exécution d'un programme ci–dessous.



Hiérarchie d'exécution d'un programme

Lors de son exécution, un programme traverse, de façon plus ou moins parallèle, la hiérarchie matérielle et logicielle. Plus l'on s'élève dans la hiérarchie matérielle, plus les éléments sont rares et coûteux. La *transition* d'un niveau à l'autre prend du temps, d'autant que, pour chaque ressource, le programme se trouve en concurrence avec d'autres. Pour analyser la dynamique d'un programme, il faut en appréhender le fonctionnement à chaque niveau hiérarchique.

Hiérarchie matérielle

En général, le temps requis pour passer d'un niveau matériel à l'autre correspond approximativement au *délai de latence* de niveau inférieur – laps de temps entre l'émission d'une requête et la réception des premières données.

Disques fixes

Lors de l'exécution d'un programme, l'opération la plus coûteuse en temps (hormis une entrée manuelle au clavier) est la récupération de code ou de données stockés sur disque :

- le contrôleur de disque doit savoir à quels blocs accéder (temps d'attente en file) ;
- le bras du disque doit trouver le cyclindre concerné (temps de recherche) ;
- les têtes de lecture et d'écriture doivent attendre le positionnement et la rotation des blocs recherchés (délai de rotation) ;
- les données doivent être transmises au contrôleur (temps de transmission) et transférées au programme d'application (temps de traitement des interruptions).

Les opérations sur disque ont des origines diverses qui ne se limitent pas aux demandes explicites d'E/S effectuées par un programme. Une grande part de l'optimisation du système consiste souvent à éliminer les E/S disque inutiles.

Mémoire réelle

La RAM est plus rapide que le disque, mais elle consomme davantage par octet. Les systèmes d'exploitation tentent de conserver dans la RAM les codes et données fréquemment utilisés et de transférer les données en sus sur le disque (ou du moins de ne pas les transférer initialement sur la RAM).

Cependant, la RAM n'est pas nécessairement plus rapide que le processeur. Sur un ESCALA, un délai de latence RAM de plusieurs cycles processeurs s'écoule entre le

moment où le matériel détecte la nécessité d'un accès RAM et celui où les données ou les instructions sont mises à disposition du processeur.

Si l'accès se fait sur une page de mémoire virtuelle stockée qui a été sur disque (ou destinée à l'être), un *défaut de page* se produit et le programme est suspendu jusqu'à ce que la page soit lue depuis le disque.

Tampons TLB (Translation Lookaside Buffer)

La mémoire virtuelle est l'un des moyens permettant de libérer les programmeurs de tout souci quant aux limites physiques du système. Ils peuvent concevoir et coder des programmes sans se préoccuper de la capacité mémoire effective : c'est le système qui se charge de traduire les adresses virtuelles des données et instructions du programme en adresses réelles, requises pour récupérer ces données et instructions en RAM. Pour limiter ces opérations de traduction, coûteuses en temps, les adresses réelles des pages de mémoire virtuelle récemment sollicitées sont conservées dans une mémoire cache appelée tampon TLB (Translation Lookaside Buffer). Ainsi, tant que le programme sollicite un même jeu réduit de pages de données et de programmes, il est inutile de retraduire les adresses à chaque accès à la RAM. En revanche, si le système tente d'accéder à une page de mémoire virtuelle qui ne bénéficie pas d'une entrée TLB (*absence en TLB*), des dizaines de cycles processeurs (*attente pour absence en TLB*) sont généralement nécessaires pour traduire les adresses.

Mémoires cache

Pour limiter la fréquence des délais de latence RAM, les ESCALA sont équipés de mémoires cache destinées aux instructions et aux données. Si l'instruction ou les données requises se trouvent déjà en mémoire cache (*présence en cache*), elles sont mises à disposition du processeur dès le cycle suivant (sans délai d'attente) ; sinon *(absence en cache*), un temps d'attente RAM se produit.

Certains systèmes disposent de deux niveaux de mémoire cache, appelés généralement L1 et L2. Si une recherche est infructueuse sur le niveau L1, elle se poursuit sur le niveau L2. En cas de nouvel échec, la recherche est reportée sur la RAM.

Sur les ESCALA, la taille et la structure des mémoires cache varient en fontion du modèle mais leur principe d'exploitation est identique. Reportez–vous à l'annexe C, Mémoire cache et adressage pour une présentation détaillée de l'architecture des mémoires cache et TLB - pour information ou pour optimiser un programme de niveau inférieur.

Pipeline et registres

L'architecture en pipeline, très évolutive, du ESCALA rend possible, dans certains cas, le traitement simultané de plusieurs instructions. De vastes ensembles de registres généraux et de registres à virgule flottante permettent de conserver une quantité considérable de données de programme dans les registres, au lieu de les stocker et de les recharger continuellement.

Les compilateurs d'optimisation du ESCALA sont conçus pour exploiter au mieux ces fonctionnalités. Il est conseillé de recourir aux compilateurs à chaque génération d'un programme exécutable (même court). Le manuel *Optimization and Tuning Guide for XL Fortran, XL C and XL C++* explique comment optimiser les programmes.

Hiérarchie logicielle

Lors de son exécution, un programme doit également traverser diverses étapes dans la hiérarchie logicielle.

Programmes exécutables

Lorsqu'un utilisateur demande l'exécution d'un programme, AIX accomplit un certain nombre d'opérations pour rendre opérationnel le programme exécutable sur disque. Tout d'abord, le système recherche dans les répertoires spécifiés à la variable d'environnement courante **PATH** de l'utilisateur la copie correcte du programme. Ensuite, le programme de

chargement du système (à ne pas confondre avec l'éditeur de liens **ld**) résout toutes les références externes du programme aux bibliothèques partagées.

Le système d'exploitation soumet la requête de l'utilisateur sous forme d'un *processus*, c'est–à–dire d'un ensemble de ressources, tel qu'un segment d'adresse virtuelle privée, comme le requiert tout programme exécutable.

Sous AIX version 4.1, le système d'exploitation crée en outre automatiquement une routine unique à l'intérieur de ce processus. Une *routine* est l'état d'exécution actuel d'une seule instance d'un programme. Sous AIX version 4.1, l'accès au processeur et à d'autres ressources est affecté sur la base de routines et non de processus. Un programme d'application peut créer plusieurs routines au sein d'un processus. Ces routines partagent les ressources du processus dont elles dépendent.

Enfin, le système se branche sur le point d'entrée du programme. Si la page de programme contenant ce point d'entrée n'est pas encore chargée en mémoire (cas notamment d'un programme récemment compilé, exécuté ou copié), l'interruption pour défaut de page provoque la lecture de la page.

Gestionnaires d'interruption

Lorsqu'un événement externe se produit, le système d'exploitation en est averti par une interruption de la routine en cours et le contrôle est transféré à un gestionnaire d'interruptions. Au préalable, l'essentiel de l'état matériel doit être sauvegardé pour que le contexte de la routine puisse être restauré une fois l'interruption traitée. Les gestionnaires d'interruption appelés pour la première fois subissent tous les temps d'attente liés au passage dans la hiérarchie matérielle (excepté ceux causés par des défauts de page). Sauf exécution très récente du gestionnaire d'interruption (ou programmes impliqués particulièrement économiques), il est peu probable que les codes ou les données soient conservés dans les tampons TLB ou les mémoires cache.

Lorsque la routine interrompue est relancée, son contexte d'exécution (contenu des registres par exemple) est *logiquement* restauré pour que la routine fonctionne normalement. En revanche, le contenu des tampons TLB et des mémoires cache doit être reconstitué à partir des demandes de programme qui ont suivi. Le gestionnaire d'interruption et la routine d'exécution subissent généralement des temps d'attente importants du fait des absences en mémoire cache et en TLB résultant de l'interruption.

Routines en attente

Chaque fois qu'une requête soumise par un programme ne peut être satisfaite immédiatement (par exemple, opération d'E/S explicite ou consécutive à un défaut de page), la routine est mise en attente jusqu'à l'exécution de la requête. Généralement, des temps d'attente dans les tampons TLB et en mémoire cache s'ajoutent au temps requis pour l'exécution de la requête.

Routines diffusables

Une routine diffusable mais non encore exécutée est plus préjudiciable qu'utile. En effet, si parallèlement, des routines sont *en cours* d'exécution, les *lignes de mémoire cache* (zones du cache contenant les instructions et/ou les données de la routine – voir annexe C : "Mémoire cache et adressage") peuvent être réutilisées et les pages de mémoire virtuelle appelées, retardant davantage la diffusion de la routine.

Routine en cours de diffusion

Le programmateur choisit la routine dont la demande de processeur est la plus forte. (Les considérations qui affectent ce choix sont discutées dans "Performance du programmateur de CPU AIX", page 2-2.) Une fois la routine diffusée, l'état logique du processeur au moment de l'interruption de la routine est restauré.

Instructions courantes

Sur les ESCALA, la plupart des instructions machine sont à même d'exécuter un cycle processeur unique, *à condition* de ne pas rencontrer d'absence en cache ou en TLB. En

revanche, si un programme se branche rapidement sur différentes zones de l'exécutable et/ou accède à des données issues de zones très diverses (générant un taux d'absence en cache et TLB élevé), le nombre *moyen* de cycles processeur par instruction exécutée peut être bien supérieur à un. On dit alors que le programme montre un faible "regroupement référentiel". Il peut utiliser le nombre minimum *d'instructions* nécessaires pour accomplir le travail mais consommer un nombre excessif de *cycles*. Cette disparité entre le nombre d'instructions et le nombre de cycles explique en partie qu'il n'est plus possible d'obtenir directement une valeur de temps en calculant simplement la "longueur du parcours" sur le listing du programme. Si un parcours réduit est *généralement* plus rapide qu'un long parcours, le rapport de vitesse peut différer sensiblement du rapport de longueur de parcours.

Les compilateurs XL réorganisent les codes selon des mécanismes très sophistiqués pour réduire le nombre de cycles nécessaires à l'exécution du programme. Le premier souci du programmeur désireux d'optimiser les performances doit être de fournir au compilateur toutes les informations requises et non de tenter de deviner et d'appliquer ses techniques. (Reportez–vous à "Utilisation des préprocesseurs et des compilateurs XL", page 4-14.) Le seul moyen de vérifier l'efficacité de l'optimisation est de mesurer les performances d'une charge de travail réelle.

Dynamique système

Créer des programmes individuels les plus efficaces possible ne suffit pas. Généralement, la personne responsable des performances n'a pas participé à la création des programmes effectivement exécutés. En outre, la plupart des niveaux de la hiérarchie décrits précédemment sont gérés par une ou plusieurs parties d'AIX. Dans tous les cas, dès lors que des programmes d'application ont été acquis et optimisés au mieux, le seul moyen d'améliorer les performances globales du système est d'intervenir au niveau du système. Les principaux éléments concernés sont :

Disques fixes	Le gestionnaire de volume logique (LVM) contrôle l'emplacement des systèmes de fichiers et des espaces de pagination sur le disque, or l'emplacement peut avoir un impact important sur le temps de recherche nécessaire au système. Les pilotes d'unité de disque contrôlent l'ordre de traitement des requêtes d'E/S.
Mémoire réelle	Le gestionnaire de mémoire virtuelle (VMM) contrôle le pool des trames de mémoire virtuelle disponibles et détermine quand et à qui voler des trames pour réalimenter le pool.
Routine en cours	Le programmateur détermine l'entité diffusable qui prendra le contrôle ensuite. (Sous AIX version 4.1, l'entité diffusable n'est plus un processus mais une routine. Reportez–vous à "Prise en charge des routines sous AIX version 4.1", page 2-2).
E/S de communication	Selon le type de charge et de liaison de communication, il peut être nécessaire de moduler la configuration des pilotes d'unité de communication, TCP/IP ou NFS.

Classes de charge de travail

Les charges de travail se répartissent généralement en un nombre limité de classes. Les types ci–dessous servent le plus souvent à classifier des *systèmes*. Mais, un seul système étant généralement sollicité pour traiter plusieurs classes, le terme "charge de travail" semble plus approprié dans un contexte de performance.

Station de travail	Charge de travail constituée d'un seul utilisateur qui soumet des travaux via le clavier natif du système et en reçoit les résultats sur la console native. En règle générale, l'objectif de performance prioritaire est, dans ce cas, un temps de réponse minimal pour le demandeur.
Multi–utilisateur	Charge de travail constituée de plusieurs utilisateurs qui soumettent des travaux via leurs terminaux respectifs. En règle générale, l'objectif de performance prioritaire est, dans ce cas, un débit maximum avec un temps de réponse assorti d'un seuil maximal, ou un temps de réponse optimal pour une charge de travail relativement constante.
Serveur	Charge de travail constituée de requêtes issues d'autres systèmes. Par exemple, la charge de travail d'un serveur de fichiers se compose en majorité de requêtes de lecture/écriture sur disque. Il s'agit, en substance, du composant E/S disque d'une charge multiutilisateur (avec en sus l'activité NFS ou DFS), le même objectif de débit maximal avec un temps de réponse limite s'applique. Les charges de serveur peuvent également se composer de programmes de calcul intensifs, transactions de bases de données, travaux d'impression, etc.

Lorsqu'un seul système traite des charges de travail de plusieurs types, utilisateurs et analyste des performances doivent s'être préalablement concertés sur les priorités relatives des objectifs en cas de conflit entre diverses charges de travail.

Procédure d'optimisation

L'optimisation est avant tout une question de gestion des ressources et de configuration du système. Voici les étapes de la procédure préconisée :

1. Identification des charges de travail sur le système.
2. Définition des objectifs :
   a. Détermination de la méthode d'évaluation des résultats.
   b. Quantification et hiérarchisation des objectifs.
3. Identification des ressources critiques limitant les performances du système.
4. Réduction des contraintes de ressources critiques :
   a. Utilisation de la ressource la plus appropriée, si le choix est possible.
   b. Réduction des contraintes de ressources critiques au niveau des programmes et des fonctions système.
   c. Utilisation parallèle des ressources.
5. Modification de l'affectation des ressources en fonction des priorités :
   a. Modification de la priorité ou des limites de ressources au niveau des programmes.
   b. Modification de la configuration des paramètres de gestion des ressources système.
6. Répétition des étapes 3 à 5 jusqu'à ce que les objectifs soient atteints (ou les ressources saturées).
7. Mise en œuvre de ressources supplémentaires, le cas échéant.

Identification des charges de travail

Identification des charges de travail Il est essentiel d'identifier *tous* les travaux exécutés sur le système. Sur les systèmes connectés à un réseau local notamment, un ensemble complexe de systèmes de fichiers montés en croisements peut facilement se développer, avec un simple accord informel des utilisateurs des systèmes. Ces éléments doivent être identifiés et pris en compte lors de l'optimisation.

Dans le cas de charges multiutilisateur, l'analyste doit quantifier le volume moyen et maximal des requêtes. Il est essentiel de faire une évaluation réaliste de la durée d'utilisation effective du terminal par un utilisateur.

Il convient de déterminer le système sur lequel effectuer les mesures et l'optimisation : système de production ou autre (ou hors des périodes d'exploitation) avec une version simulée de la charge effective. Il incombe à l'analyste de choisir entre, d'une part, les résultats issus d'un environnement de production – par nature, plus proches de la réalité – et, d'autre part, la productivité maximale réalisable dans un environnement non productif (conditions permettant de pratiquer des tests sans risquer de nuire aux performances du système).

Définition des objectifs

Les objectifs doivent être définis quantitativement, même si le résultat souhaité est souvent exprimé en termes subjectifs (par exemple, un temps de réponse "satisfaisant"). De plus, l'analyste doit viser à améliorer ce qui est important même s'il est plus tentant d'intervenir sur ce qui est mesurable. Si aucune mesure fournie par le système ne correspond aux améliorations recherchées, il vous incombe d'en fixer une.

Quantifier les objectifs a certes pour finalité d'orienter l'optimisation vers la réalisation des valeurs fixées, mais a surtout un rôle essentiel quant à la liberté d'action de l'analyste. Il s'agit en effet de définir, en concertation avec les utilisateurs, les priorités à respecter. Faute de cette entente préalable, l'analyste se verra sans cesse contraint de consulter les

intéressés, de justifier les mesures qu'il prend, etc. Perte de temps et d'énergie : l'optimisation du travail de l'analyste laisserait beaucoup à désirer. Si l'exploitation et la prise en charge du système sortent du cadre de l'entreprise, il convient de recourir à un accord de service écrit entre utilisateurs et fournisseurs pour s'assurer que les objectifs de performance et les priorités ont été clairement compris.

Identification des ressources critiques

En général, les performances d'une charge donnée sont déterminées par la disponibilité et la rapidité d'une ou deux ressources système critiques. L'analyste se doit de les identifier précisément pour ne pas s'enfermer dans une boucle sans fin de tests et d'erreurs.

Les systèmes sont dotés de ressources réelles et logiques. Les ressources critiques réelles sont généralement plus faciles à identifier, grâce aux nombreux outils de performances système disponibles qui en testent l'utilisation. Voici les ressources réelles dont l'impact sur les performances est le plus fréquent :

- cycles CPU,
- mémoire,
- bus d'E/S,
- cartes diverses,
- bras de disque,
- espace disque,
- accès réseau.

L'identification des ressources logiques est moins évidente. Il s'agit généralement d'abstractions de programmation qui partitionnent les ressources réelles. Le partitionnement permet le partage et la gestion des ressources réelles.

Voici des exemples de ressources réelles partionnées en ressources logiques :

CPU

- Tranche horaire CPU

Mémoire

- Trames de page
- Piles
- Tampons
- Files d'attentes
- Tables
- Verrous et sémaphores

Espace disque

- Volumes logiques
- Systèmes de fichiers
- Fichiers
- Partitions

Accès réseau

- Paquets
- Canaux

Il est essentiel de tenir compte des deux types de ressources. En effet, des routines peuvent être bloquées faute de ressources logiques comme de ressources réelles.

Augmenter la ressource réelle impliquée ne garantit pas la création de ressources logiques supplémentaires. Prenons, par exemple, un démon d'E/S de bloc NFS (**biod**, reportez-vous à "Optimisation de NFS", page 9-36). Un **biod** est requis sur le client pour traiter chaque requête NFS d'E/S distante en attente. Le nombre d'opérations d'E/S NFS simultanées est donc limité par le nombre de **biod**. En cas de pénurie de **biod**, les indicateurs de mesure du système signalent que les liaisons CPU et de communications sont relativement peu utilisées. Vous pouvez ainsi avoir l'impression fausse que le système est sous-utilisé (et trop lent). Or, il s'agit simplement d'un manque de **biod** qui limite l'activité du reste des ressources. Un **biod** utilise des cycles processeur et de la mémoire, mais vous ne pouvez résoudre ce problème en ajoutant simplement de la mémoire réelle ou en passant à une CPU plus rapide. La solution consiste à créer davantage de ressources logiques (**biod**).

Des ressources logiques et des goulots d'étranglement peuvent être créés par inadvertance pendant la développement d'applications. Une méthode de transmission des données ou de contrôle d'une unité peut en effet être à l'origine de la création d'une ressource logique. Dans ce cas, vous ne disposez généralement d'aucun outil pour gérer l'utilisation des ressources créées ni d'interface pour contrôler leur affectation. Par ailleurs, leur existence n'est souvent révélée que lorsque survient un problème spécifique.

Réduction des contraintes de ressources critiques

Choix de la ressource

Privilégier une ressource doit être le fruit d'une analyse minutieuse guidée par des objectifs précis. Par exemple, pour le développement d'une application, il peut être décidé d'opter pour une consommation CPU mininale, au détriment de la consommation de mémoire. De même, lors de la configuration du système, il faut décider du lieu de stockage des fichiers : localement sur une station de travail ou à distance sur un serveur.

Réduction des contraintes

Les applications développées localement doivent être étudiées pour déterminer la possibilité d'augmenter l'efficacité d'une fonction ou de supprimer des fonctions inutiles. Au niveau de la gestion du système, il peut être envisagé de transférer sur d'autres systèmes ou de différer les charges peu prioritaires demandant l'accès à des ressources critiques.

Utilisation parallèle des ressources

Pour leur exécution, les charges de travail requièrent plusieurs ressources système. Il faut donc exploiter le fait que celles-ci soient séparées et utilisables en parallèle. Par exemple, les algorithmes de lecture anticipée sous AIX détectent qu'un programme accède à un fichier séquentiellement et planifient la lecture parallèlement au traitement par l'application des données précédentes. Ce traitement en parallèle s'applique également à la gestion du système. Par exemple, si une application accède à plusieurs fichiers simultanément, ajouter une unité de disque et répartir les fichiers concernés sur plusieurs disques ne peut qu'améliorer le taux d'E/S disque.

Modification de l'affectation des ressources en fonction des priorités

AIX offre plusieurs méthodes pour affecter des niveaux de priorités aux activités. Certaines, telles que la régulation disque, sont définies au niveau système. D'autres, telles que la priorité des processus, peuvent être définies par chaque utilisateur en fonction de l'importance attribuée à une tâche donnée.

Répétition des étapes d'optimisation

Dans l'analyse des performances, il faut toujours partir du principe qu'un autre goulot d'étranglement va se produire. Diminuer l'exploitation d'une ressource se traduit nécessairement par la réduction du débit ou l'augmentation du temps de réponse d'une autre ressource. Prenons un exemple.

```
CPU: 90%        Disk: 70%        Memory 60%
```

La charge de ce système est concentrée sur la CPU. Si l'on régule la charge de façon à ramener le taux d'utilisation de la CPU à 45 %, il est probable que les performances soient doublées. En contrepartie, la charge est limitée au niveau des E/S, comme suit :

```
CPU: 45%        Disk: 90%        Memory 60%
```

L'amélioration côté CPU permet aux programmes de soumettre plus rapidement les requêtes de disque mais la capacité maximale de l'unité de disque est alors atteinte. L'amélioration effective des performances est en fait de l'ordre de 30 % au lieu des 100 % escomptés.

Il y a toujours une nouvelle ressource critique. La question est de savoir si les objectifs ont été atteints avec les ressources disponibles.

Mise en œuvre de ressources supplémentaires

Si toutes les méthodes précédentes n'ont pas permis d'atteindre les objectifs de performancee du système, il faut envisager d'améliorer ou d'augmenter la ressource critique. S'il s'agit d'une ressource logique et que la ressource réelle correspondante convient, la ressource logique peut être augmentée sans surcoût. S'il s'agit d'une ressource réelle, plusieurs questions se posent :

- De combien la ressource critique doit-elle être augmentée pour mettre fin au goulot d'étranglement ?
- Cette extension sera-t-elle suffisante pour atteindre les objectifs de performance du système ?
- Ou une autre ressource sera-t-elle saturée avant cela ? Dans le cas où plusieurs ressources critiques successives se présentent, est-il plus économique de les augmenter toutes ou de transférer une partie de la charge actuelle sur un autre système ?

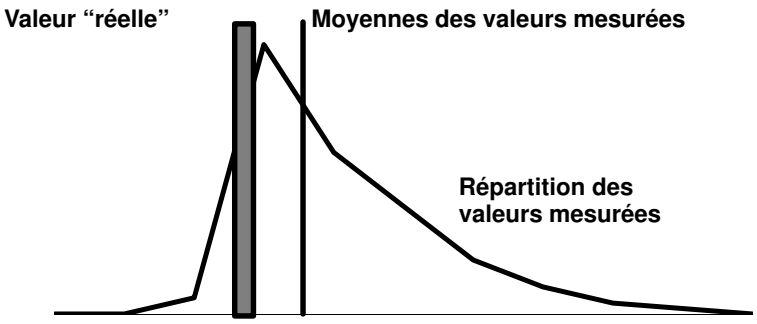
Essais comparatifs – les données de performance sont inévitablement modifiées

Lorsque l'on compare les performances d'un logiciel donné dans plusieurs environnements différents, un certain nombre d'erreurs, techniques et conceptuelles, sont possibles. La présente section est une simple mise en garde. D'autres sections de ce manuel traitent des différents moyens de mesure des temps écoulés et des temps spécifiques aux processus.

Lorsque l'on mesure le temps (mesuré sur l'horloge murale) nécessaire au traitement d'un appel système, on obtient un nombre qui constitue la somme des éléments suivants :

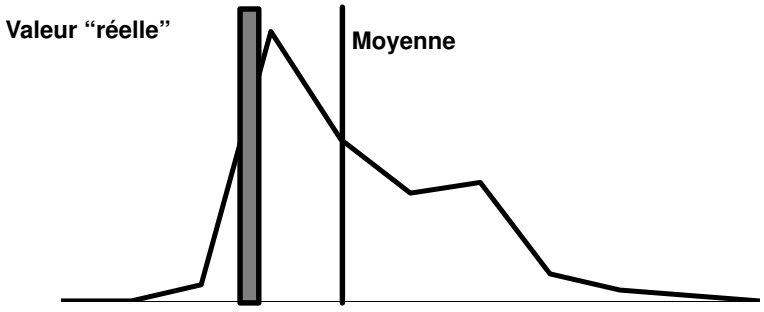
- Temps réel d'exécution des instructions de traitement du service.
- Plusieurs périodes de temps pendant lesquelles le processeur a été bloqué dans l'attente d'instructions ou de données en provenance de la mémoire (coût des absences en mémoire cache et/ou en TLB).
- Temps requis pour accéder à "l'horloge" en début et en fin d'appel.
- Temps utilisé par les événements réguliers tels que les interruptions de l'horloge système.
- Temps utilisé par les événements plus ou moins aléatoires tels que les interruptions d'entrée-sortie.

Pour minimiser les risques d'erreur, nous mesurons généralement plusieurs fois la charge de travail. Dans la mesure où tous les facteurs externes *s'ajoutent* au temps réel, l'ensemble des mesures produit généralement la courbe suivante :

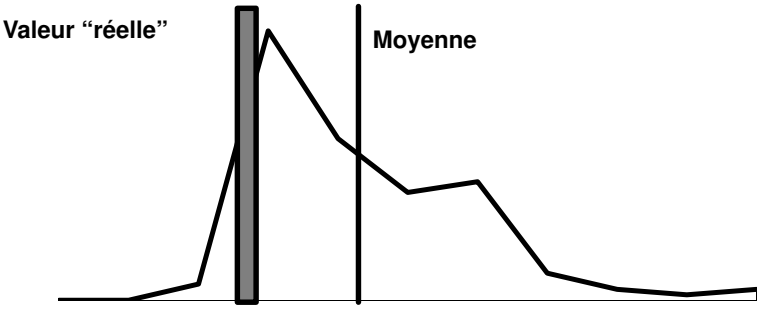


Le faible niveau de la courbe en son extrémité peut indiquer une mise en mémoire cache optimale de faible probabilité ou résulter de l'arrondissement des valeurs.

Un événement externe régulier peut entraîner la présence de deux maxima sur la courbe. Exemple :



Une ou deux interruptions longues peuvent modifier encore davantage le tracé de la courbe :



La répartition des mesures autour de la valeur ”réelle” n’est pas aléatoire. Il convient donc d’appliquer avec précaution les tests classiques de statistiques par déduction. En outre, dans certains cas, ni la moyenne, ni la valeur ”réelle” ne caractérisent correctement les performances.

Voir aussi

CPU

- ”Performances du programmateur de CPU sous AIX” : traite de la gestion de la CPU sous AIX.
- ”Contrôle et optimisation de la CPU” : présente les outils et techniques de gestion de la CPU.

Mémoire

- ”Performances du gestionnaire de mémoire virtuelle (VMM)” : décrit l’architecture de gestion de mémoire AIX.
- ”Contrôle et optimisation de la mémoire” : présente les outils et techniques de gestion de la mémoire.

Disques

- ”Gestion AIX du stockage sur disque fixe” : décrit la structure du support disque fixe sous AIX.
- ”Préinstallation du disque” : traite de la planification des informations sur les performances relatives des disques fixes de ESCALA.
- ”Contrôle et optimisation des E/S disque” : traite du contrôle de la réorganisation et de l’extension du stockage sur disque.

Chapitre 2. Gestion des ressources AIX

Ce chapitre décrit les composants AIX de gestion des ressources ayant le plus d'impact sur les performances du système et indique comment les optimiser. Pour des conseils d'optimisation sur une ressource spécifique, reportez-vous au chapitre correspondant.

- "Performances du programmateur CPU sous AIX"
- "Performances du gestionnaire de mémoire virtuelle (VMM)"
- "Performances de gestion AIX du stockage sur disque fixe"

Pour des conseils d'optimisation spécifiques, reportez-vous à :

- "Contrôle et optimisation de la CPU"
- "Contrôle et optimisation de la mémoire"
- "Contrôle et optimisation des E/S disque"

Performances du programmateur CPU sous AIX

Le programmateur CPU a été largement remanié du fait de la nouvelle prise en charge des routines sous AIX version 4.1. D'un point de vue conceptuel, l'algorithme de planification et la politique d'attribution des priorités n'ont pas changé par rapport à la version 3.2.5, mais dans le détail, nombre de modifications ont été apportées. Pour les applications non modifiées et exécutées sur des monoprocesseurs, l'impact net des modifications est très faible. Cependant, toute personne concernée par l'optimisation des performances doit être à même de comprendre les modifications et les nouvelles opportunités qui sont offertes.

Prise en charge des routines sous AIX version 4.1

Une *routine* est comparable à un processus système de niveau inférieur : il s'agit d'une entité diffusable qui exige pour sa création moins de ressources qu'un processus AIX. Dans le programmateur d'AIX version 4.1, la routine constitue l'entité diffusable de base.

Cela ne signifie pas que les routines ont entièrement supplanté les processus. En fait, les charges de travail qui ont subi une migration directement à partir de versions précédentes d'AIX créent des processus comme auparavant. Chaque nouveau processus est créé avec une seule routine qui est en concurrence avec les routines des autres processus pour l'accès aux ressources CPU. Si le processus possède les ressources utilisées lors de l'exécution, la routine, quant à elle, ne possède que son état courant.

Lorsque des applications nouvelles ou modifiées créent des routines supplémentaires à la faveur du nouveau support de routines d'AIX, cette création a lieu dans le cadre du processus. Les ressources partagent de ce fait le segment privé et les autres ressources du processus.

Une routine utilisateur dans un processus est dotée d'une *portée concurrentielle*. Si la portée concurrentielle est *globale*, la routine est en conccurrence avec toutes les autres routines du système pour l'accès aux ressources CPU. (La routine créée en même temps qu'un processus est dotée d'une portée concurrentielle globale.) Si la portée est *locale*, la routine est en concurrence uniquement avec les routines de son processus pour le partage du temps CPU accordé aux processus.

L'algorithme qui permet de désigner la routine suivante à exécuter est appelé politique de planification.

Planification des routines de portée concurrentielle globale ou locale

Trois politiques de planification sont disponibles sous AIX version 4 :

FIFO	La routine est exécutée jusqu'à son terme, sauf si un blocage se produit. Elle utilise la CPU ou une routine dotée d'une priorité supérieure devient diffusable. Seules les routines à priorité fixe peuvent être exécutées en mode FIFO.
RR	Cette période est définie selon le même mécanisme de répartition par permutation circulaire (RR) que celui utilisé sous AIX version 3 (répartition par tranches horaires de 10 ms). La routine se place à la fin de la file d'attente des routines diffusables de même priorité lorsqu'elle s'exécute au bout d'une période de temps spécifiée. Seules les routines à priorité fixe peuvent être exécutées en mode RR.
OTHER	Ce mode est "défini par implantation" selon POSIX1003.4a. Sous AIX version 4.1, le mode OTHER est déclaré équivalent au mode RR, à ceci près qu'il s'applique aux routines à priorité variable. Dans ce cas, la priorité de la routine en cours est recalculée à chaque interruption d'horloge si bien que la routine peut être amenée à passer le contrôle à une autre routine diffusable. C'est le mécanisme appliqué sous AIX version 3.

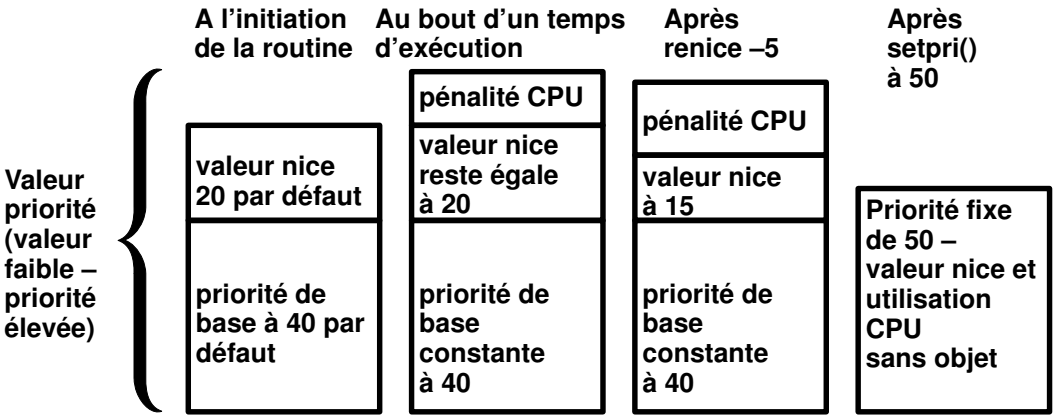
Les routines sont particulièrement utiles pour les applications composées de plusieurs processus asynchrones : ces applications peuvent soumettre au système une moindre charge dès lors qu'elles sont converties en structures multiroutines.

Priorité des processus et des routines

Les outils de gestion des priorités disponibles sous la version 3.2.5 manipulent les priorités au niveau des processus. Sous AIX version 4.1, la priorité d'un processus ne fait qu'annoncer celle de la routine. Appeler **fork()** entraîne la création d'un processus contenant une routine. C'est la routine qui reçoit le niveau de priorité auparavant attribué au processus sous la version 3.2.5 d'AIX. La présente description est applicable aux deux versions.

Le noyau tient à jour pour chaque routine une *valeur de priorité* (aussi appelée *priorité de planification*). La priorité est définie par un entier positif, inversement proportionnel à l'importance de la routine associée. Autrement dit, une valeur de priorité faible désigne une routine importante. Lorsque le programmateur cherche la routine suivante à diffuser, il choisit celle dont la priorité a la valeur la plus basse.

Une priorité peut être *fixe* ou *variable*. Une priorité fixe a une valeur constante tandis que la priorité d'une routine à priorité variable est la somme de la priorité minimale des routines utilisateur (constante 40), de la valeur *nice* (20 par défaut ou valeur définie par la commande **nice** ou **renice**) et de la pénalité d'utilisation CPU. La figure "Evolution de la valeur de priorité" illustre cette évolution.



Evaluation de la valeur de priorité

La valeur nice d'une routine, définie lors de la création de la routine, est constante pendant toute la durée de vie de la routine, sauf changement explicite par un utilisateur via la commande **renice** ou les appels système **setpri**, **setpriority** ou **nice**.

La pénalité CPU est un nombre entier calculé d'après l'utilisation récente de la CPU par une routine. Cette valeur est incrémentée de 1 chaque fois que la routine utilise la CPU pendant une tranche horaire (10 ms), jusqu'à une valeur maximale de 120. Une fois par seconde, les valeurs d'utilisation récente de la CPU de toutes les routines sont réduites. Le résultat obtenu est :

- La priorité d'une routine à priorité variable décroît au fur et à mesure que l'utilisation récente de la CPU s'accroît, et inversement. Il en découle que, en moyenne, plus une routine a récemment bénéficié de tranches horaires, moins elle a de chance d'être d'affectée à la tranche horaire suivante.
- La priorité d'une routine à priorité variable décroît au fur et à mesure que la valeur nice s'accroît, et inversement.

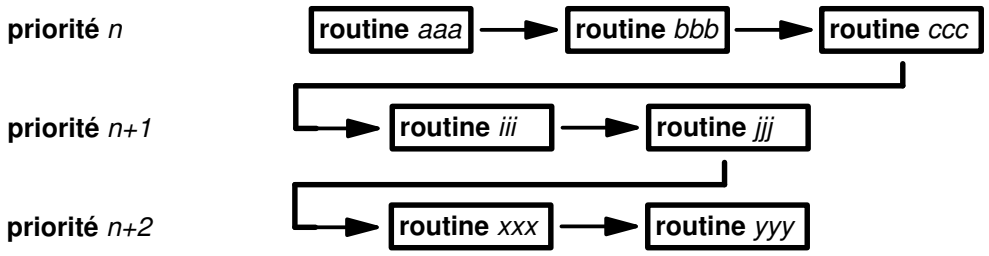
La priorité d'une routine peut être fixée via la sous-routine **setpri**. La commande **ps** permet d'afficher la priorité, la valeur nice et les valeurs d'utilisation CPU à court terme d'un processus.

Pour en savoir plus sur les commandes **nice** et **renice**, reportez-vous à "Contrôle des conflits pour CPU", page 6-20.

Pour en savoir plus sur le calcul de la pénalité CPU et la diminution des valeurs d'utilisation de la CPU à court terme, reportez vous à "Optimisation du calcul des priorités d'un processus via **schedtune**, page 6-23.

File d'attente d'exécution du programmateur d'AIX

Le programmateur maintient une *file d'attente d'exécution* de toutes les routines prêtes à être diffusées. La figure "File d'attente d'exécution" en est une illustration symbolique.



File d'attente d'exécution

Les routines diffusables d'un niveau de priorité donné se suivent dans la file d'attente d'exécution.

Lorsqu'une routine est placée "en fin de file d'attente d'exécution" (par exemple, lorsqu'elle s'exécute à la fin d'une tranche horaire), elle est positionnée après la dernière routine en attente dotée de la même priorité.

Tranche horaire CPU

La tranche horaire CPU est l'intervalle entre deux recalculs de la priorité. En temps normal, ce calcul est effectué à chaque impulsion de l'horloge système, c'est-à-dire toutes les 10 millisecondes. L'option **–t** de la commande **schedtune** (voir page A-6) permet d'augmenter cet intervalle par incréments de 10 millisecondes. Rappelons qu'une tranche horaire ne correspond pas à une quantité *garantie* de temps processeur. Il s'agit de la durée *maximale* pendant laquelle une routine peut s'exécuter sans risquer d'être remplacée par une autre. Il existe plusieurs cas de figure où une routine peut être supplantée par une autre avant de bénéficier d'une tranche horaire complète.

Voir aussi

"Gestion des ressources AIX"

"Contrôle et optimisation de la CPU"

Commandes **nice**, **ps** et **renice**

Sous-routine **setpri**

Sous-routines **getpriority, setpriority** et **nice**.

Performances du gestionnaire de mémoire virtuelle (VMM)

L'espace d'adresse virtuelle du ESCALA est partitionné en segments (pour une description détaillée de la structure d'adressage virtuelle, reportez vous à l'annexe C, "Mémoire cache et adressage"). Un segment est une portion contiguë de 256 Mo de l'espace d'adresse virtuelle dans laquelle un objet de données peut être mappé. L'adressage processus-données est géré au niveau du segment (ou objet) de sorte qu'un segment peut être partagé par des processus ou rester privé. Par exemple, des processus peuvent partager des segments de codes, mais disposer de segments de données privés distincts.

Le gestionnaire de mémoire virtuelle (VMM) est décrit dans les sections suivantes :

- "Gestion de la mémoire réelle"
- "Utilitaire de contrôle de l'occupation mémoire"
- "Affectation et réclamation d'emplacements dans l'espace de pagination"

Gestion de la mémoire réelle

Les segments de mémoire virtuelle sont partitionnés en unités de taille fixe appelées *pages*. La taille d'une page sous AIX est de 4 096 octets. Chaque page d'un segment peut être implantée dans la mémoire réelle (RAM) ou stockée sur disque tant qu'elle n'est pas utilisée. De même, la mémoire réelle est divisée en *trames de page* de 4 096 octets. Le rôle de VMM est de gérer l'affectation des trames de page de mémoire virtuelle et de résoudre les références dans le programme aux pages de mémoire virtuelle qui ne se trouvent pas en mémoire réelle ou n'existent pas encore (c'est par exemple le cas lorsqu'un processus fait référence pour la première fois à une page de son segment de données).

La quantité de mémoire virtuelle utilisée à un instant donné pouvant être supérieure à la capacité de la mémoire réelle, VMM se charge de stocker le surplus sur disque. En terme de performances, VMM doit remplir deux objectifs, quelque peu contradictoires :

- Réduire le coût global en temps processeur et largeur de bande induit par l'utilisation de la mémoire virtuelle.
- Réduire le coût en temps de réponse des défauts de page.

Dans cette optique, le gestionnaire VMM tient à jour une *liste des disponibilités* de trames de page en cas de défaut de page. Il utilise un *algorithme* de *repositionnement de page* pour déterminer les pages de mémoire virtuelle actuellement en mémoire dont les trames doivent être réaffectées à la liste des disponibilités. Le mécanisme de l'algorithme de repositionnement de page est le suivant :

- Les segments de mémoire virtuelle sont répartis en segments persistants et segments de travail.
- Les segments de mémoire virtuelle sont répartis en deux autres catégories selon qu'ils contiennent de la mémoire de calcul ou de la mémoire de fichier.
- Les pages de mémoire virtuelle à l'origine d'un défaut de page sont recherchées.
- Parmi les défauts de page, l'algorithme distingue les défauts de nouvelle page initiale et les défauts de page déjà référencée.
- Les statistiques sont tenues à jour sur la base des défauts de pages déjà référencées au niveau de chaque segment de mémoire virtuelle.
- Les limites inférieures définies par l'utilisateur sont prises en compte par l'algorithme.

Le mécanisme de l'algorithme et du système de liste des disponibilités est détaillé ci-après.

Liste des disponibilités

Le gestionnaire VMM tient à jour la liste des trames de pages disponibles pour faire face aux défauts de page. Dans la plupart des environnements, VMM doit mettre à jour cette liste en réaffectant les trames de page des processus en cours. C'est l'algorithme de repositionnement de page de VMM qui décide des pages de mémoire virtuelle dont les trames doivent être réaffectées. Le nombre de trames concernées dépend des seuils définis dans VMM.

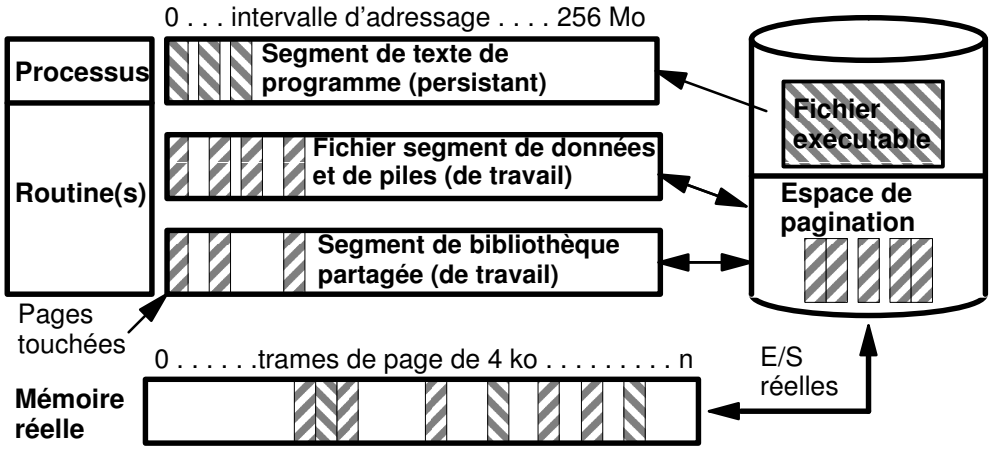
Sous AIX version 3, le contenu des trames de page réaffectées dans la liste des disponibilités n'est pas perdu. Si une page de mémoire virtuelle référencée avant la trame qu'elle occupe est utilisée pour compenser un défaut de page, la trame en question est supprimée de la liste et réaffectée au processus défaillant. Ce phénomène est appelé une *réclamation*. Les réclamations ne sont pas prises en charge par AIX version 4.1.

Segments persistants et segments de travail

Les pages d'un *segment persistant* bénéficient d'emplacements de stockage permanents sur disque. Les fichiers contenant des données ou des exécutables sont mappés à des segments persistants. Ainsi, lorsqu'une telle page est modifiée et ne peut plus être conservée en mémoire réelle, VMM la réinstalle à l'emplacement qui lui est réservé. Si la page n'a pas été modifiée, sa trame est simplement réaffectée dans la liste des disponibilités. Enfin, si elle est de nouveau référencée, VMM en crée une copie à partir de l'emplacement permanent.

Les *segments de travail* sont des segments temporaires qui ne bénéficient pas d'emplacements de stockage permanents et dont la durée de vie est limitée à leur temps d'utilisation par un processus. Les piles de processus et les zones de données sont mappées à ces segments de travail, de même que les segments de texte du noyau et de l'extension du noyau, ainsi que les segments de données et de texte des bibliothèques partagées. Lorsqu'elles ne peuvent être conservées en mémoire réelle, les pages des segments de travail sont également stockées sur le disque. Le stockage a lieu dans l'espace de pagination du disque.

La figure "Segments de stckockage de travail et persistants" illustre les relations entre certains types de segment et l'emplacement de leurs pages sur le disque. Elle indique également les emplacements réels (arbitraires) des pages se trouvant en mémoire réelle.



Segments persistants et segments de travail

Il existe différents types de segments persistants. *Les segments client* sont utilisés pour mapper les fichiers distants (tels que les fichiers accessibles via NFS) parmi lesquels les fichiers exécutables. Les pages des segments client sont sauvegardées et restaurées via le réseau, à leur emplacement de fichier permanent et non dans l'espace de pagination du disque local. Les segments journalisés et différés sont des segments persistants qui doivent être mis à jour de façon atomique. Si une page d'un segment journalisé ou différé est sélectionnée pour être ôtée de la mémoire réelle (*déchargée*), elle doit être copiée dans

l'espace de pagination du disque (à moins que son état n'autorise sa validation, c'est-à-dire son inscription à l'emplacement de fichier permanent).

Mémoire de calcul et mémoire de fichier

La *mémoire de calcul* est constituée des pages appartenant aux segments de stockage de travail ou aux segments de texte de programme. (Un segment est considéré comme texte de programme si une absence en cache d'instructions se produit sur n'importe laquelle de ses pages.) *La mémoire de fichier* est constituée par les autres pages.

Repagination

Il existe deux catégories de défaut de page : *défaut de nouvelle page* et *défaut de page déjà référencée*. Les défauts qui relèvent de la première catégorie se produisent lorsqu'aucun article de la page concernée n'a été référencé récemment. Ceux de la seconde catégorie concernent les pages déjà référencées qui le sont à nouveau, mais ne se trouvent pas en mémoire car elles ont été déplacées depuis (éventuellement sur disque). Une politique idéale de repositionnement de page serait d'éliminer entièrement les défauts de page déjà référencée (en admettant que la mémoire réelle soit suffisante) par le vol systématique des trames des pages qui ne sont pas appelées à être de nouveau référencées. Ainsi, le nombre de ces défauts de page est inversement proportionnel à l'efficacité de l'algorithme de repositionnement, du fait du vol continu des pages en mémoire (réduisant d'autant le volume global des E/S au profit des performances du système).

Pour estimer la catégorie du défaut de page, VMM tient un *tampon historique des pages référencées* où sont consignés les ID des pages ayant fait l'objet des N défauts de page les plus récents (N étant le nombre maximum de trames que la mémoire peut contenir). Par exemple, une mémoire de 16 Mo requiert un tampon de 4 096 entrées pour l'historique des pages référencées. Si l'ID de la page en entrée figure déjà dans le tampon, la page est considérée comme déjà référencée. VMM évalue également les taux de défauts de page référencée en mémoire de calcul et en mémoire de fichier séparément, en tenant un compte distinct pour chaque type de mémoire. Ces taux sont multipliés pas 0,9 à chaque exécution de l'algorithme pour donner une image de l'activité récente de repagination plus fidèle que l'historique.

Seuils VMM

Plusieurs seuils numériques définissent les objectifs de VMM. Lorsqu'un de ces seuils est dépassé, VMM prend les mesures nécessaires pour ramener l'état de la mémoire à l'intérieur des limites définies. Cette section traite des seuils modifiables par l'administrateur système via la commande **vmtune**.

Le nombre de trames de page figurant dans la liste des disponibilités est contrôlé par les paramètres :

minfree Nombre minimum de trames de page mémoire réelle admissibles dans la liste des disponibilités. Lorsque la taille de la liste des disponibilités passe en dessous de ce seuil, VMM commence à voler des pages. Il continue à en voler jusqu'à ce que la taille de la liste des disponibilités atteigne **maxfree**.

maxfree Taille maximale que la liste des disponibilités peut atteindre sous l'effet du vol de pages effectué par VMM. Il peut arriver que la liste dépasse cette taille, lorsque les processus se terminent et libèrent les pages des segments de travail ou que des fichiers possédant des pages en mémoire sont supprimés.

VMM tente de maintenir la liste à une taille au moins égale à **minfree**. Si, du fait des défauts de page et/ou des requêtes système, la taille descend en-deçà de **minfree**, l'algorithme de repositionnement de page s'exécute. Il est nécessaire de respecter une taille minimale (valeur **minfree**, par défaut) pour plusieurs raisons. Par exemple, l'algorithme de prérecherche séquentielle d'AIX requiert plusieurs trames simultanément pour chaque processus effectuant une lecture séquentielle. De même, VMM doit veiller à ce que

suffisamment d'espace soit disponible pour charger une page nécessaire à la libération d'une trame de page, faute de quoi un blocage du système risque de se produire.

Les seuils suivants sont exprimés en pourcentages. Ils représentent la fraction de la mémoire réelle totale occupée par des pages de fichiers (n'appartenant pas à un segment de mémoire de calcul).

minperm Si le pourcentage de mémoire réelle occupée par les pages de fichiers est inférieur à ce niveau, l'algorithme de repositionnement de page vole à la fois les pages de mémoire de calcul et de mémoire de fichier sans tenir compte des taux de repagination.

maxperm Si le pourcentage de mémoire réelle occupée par les pages de fichiers dépasse ce niveau, l'algorithme de repositionnement de page ne vole que les pages de fichiers.

Lorsque le pourcentage de mémoire réelle occupée par les pages de fichier est compris entre **minperm** et **maxperm**, VMM ne vole généralement que les pages de fichiers, mais si le taux de repagination des pages de fichiers est supérieur à celui des pages de mémoire de calcul, il vole également ces dernières.

L'algorithme de repositionnement de page s'assure que les pages de mémoire de calcul sont correctement traitées. Par exemple, la lecture séquentielle d'un fichier de données volumineux stocké en mémoire ne doit pas entraîner la perte de pages de texte d'un programme susceptibles d'être sollicitées à court terme. Par le truchement des seuils et des taux de repagination, l'algorithme contrôle le traitement des deux types de page (en accordant une légère préférence aux pages de mémoire de calcul).

Utilitaire de contrôle de l'occupation mémoire

Lorsqu'un processus fait référence à une page de mémoire virtuelle non stockée sur le disque (qu'elle ait été déchargée ou jamais chargée), la page concernée doit être chargée et, le plus souvent, une ou plusieurs pages doivent être déchargées. Ces opérations génèrent un flux d'E/S et retardent la progression du processus.

AIX tente, via l'algorithme de repositionnement de page, de voler de la mémoire réelle des pages peu susceptibles d'être référencées à court terme. Un algorithme efficace permet au système d'exploitation de conserver en mémoire suffisamment de processus actifs pour maintenir la CPU en activité. Mais, à un certain niveau de sollicitation de mémoire (fonction de la capacité mémoire totale du système, du nombre de processus, de l'évolution des besoins en mémoire de chaque processus et de l'algorithme de repositionnement de page), aucune page ne peut être déchargée sur le disque car elles sont toutes susceptibles d'être réutilisées à court terme par les processus actifs.

Dans ce cas, les pages sont continuellement chargées et déchargées. Ce phénomène est appelé *emballement*. L'emballement génère une activité incessante d'E/S sur le disque de pagination et répond dès la diffusion d'un processus par un défaut de page quasi systématique. Mais l'aspect le plus pernicieux de l'emballement est qu'il entraîne le système dans une boucle infinie alors même qu'il n'a été provoqué que par un excès bref et fortuit de la charge (par exemple, si tous les utilisateurs du système appuient sur la touche Entrée à la même seconde).

AIX dispose d'un algorithme de contrôle de la charge mémoire qui, lorsqu'il détecte un emballement du système, interrompt des processus actifs et retarde quelque temps le lancement de nouveaux processus. Cinq paramètres définissent les taux et les seuils pour l'algorithme. Les valeurs par défaut de ces paramètres ont été choisies pour s'adapter à un vaste éventail de types de charges de travail. Dans certains cas, il est possible de recourir à un mécanisme de modulation (ou de désactivation) du contrôle de charge (reportez-vous à "Optimisation du contrôle de charge mémoire VMM", page 7-14).

Algorithme de contrôle de la charge mémoire

Le mécanisme de contrôle de charge mémoire évalue, toutes les secondes, si la mémoire disponible est suffisante pour le jeu de processus actifs. Lorsqu'une *surcharge mémoire* est détectée, certains processus sont interrompus, ce qui réduit le nombre de processus actifs et le taux de surcharge de la mémoire : dès que leur état le permet, l'exécution de leurs routines est suspendue. Les pages des processus interrompus deviennent rapidement obsolètes et sont déchargées par l'algorithme de repositionnement de page. Cette opération libère suffisamment de trames de page pour permettre aux processus actifs restants de continuer. Dans l'intervalle de suspension des processus, les nouveaux processus créés sont également interrompus, empêchant toute nouvelle arrivée de travaux sur le système. Les processus interrompus ne sont réactivés qu'au bout d'un certain laps de temps - où aucun risque d'emballement ne s'est manifesté. Une fois cet *intervalle de sécurité* écoulé, les routines des processus interrompus sont progressivement réactivées.

Les paramètres de contrôle de la charge mémoire spécifient : le seuil de surcharge de la mémoire système, la durée (en secondes) de l'intervalle de sécurité, le seuil de surcharge d'un processus au-delà duquel son interruption est jugée possible, le nombre minimum de processus actifs lors de l'interruption de processus et le délai minimum (en secondes) entre la suspension d'un processus et sa réactivation.

Ces paramètres et leur valeur par défaut (entre parenthèses) sont les suivants :

h	Seuil haut de surcharge mémoire (6)
w	Attente avant la réactivation des processus suspendus (1 seconde)
p	Seuil de surcharge de la mémoire pour un *p*rocessus (4)
m	Degré *m*inimum de multiprogrammation (2)
e	Délai *é*coulé sans interruption (2 secondes)

Les valeurs de tous ces paramètres sont des entiers positifs.

Paramètre h

Le paramètre *h* définit le seuil de surcharge de la mémoire. Le système de contrôle de charge mémoire tente d'interrompre les processus lorsque ce seuil est dépassé pendant une seconde. Ce seuil est issu de deux mesures directes : le nombre de pages inscrites dans l'espace de pagination pendant la dernière seconde, et le nombre de vols de pages ayant eu lieu durant cette même dernière seconde. Le nombre d'écritures de pages est *généralement* nettement inférieur au nombre de vols de page. La mémoire est considérée surchargée lorsque :

$$\frac{\text{nombre de pages écrites lors de la dernière seconde}}{\text{nombre de pages volées lors de la dernière seconde}} > \frac{1}{h}$$

Plus ce rapport augmente, plus le risque d'emballement est grand. La valeur par défaut de *h* est 6, ce qui signifie qu'il y a risque d'emballement du système lorsque le rapport entre le nombre de pages écrites et le nombre de pages volées est supérieur à 17 %. Une valeur de *h* inférieure (pouvant aller jusqu'à 0, le test étant effectué sans effectuer une division réelle) augmente le seuil de détection du phénomène d'emballement. Autrement dit, le système est autorisé à s'approcher davantage des conditions d'emballement avant que ne se déclenche l'interruption des processus. Le rapport ci-dessus a été choisi car il est relativement peu dépendant de la configuration. Quels que soient la capacité de pagination du disque et le nombre de mégaoctets de mémoire installée sur le système, un rapport faible diminue les risques d'emballement. Un rapport avoisinnant la valeur 1 signifie un emballement certain. Tout laps de temps pendant lequel la mémoire n'est pas surchargée peut être défini comme période sûre.

Paramètre w

Le paramètre *w* contrôle le nombre d'intervalles de 1 seconde durant lesquels le rapport ci-dessus doit être inférieur à 1/*h* – avant réactivation des processus interrompus. La valeur

par défaut, 1 seconde, est proche de la valeur minimale admise (zéro). Une valeur de 1 seconde tente de réactiver les processus dès expiration du délai de sécurité de 1 seconde. Si une valeur plus élevée est attribuée à *w*, les temps de réponse pour les processus interrompus risquent d'être inutilement longs, le processeur étant inactif faute de processus.

Paramètre p

Le paramètre *p* détermine si un processus peut être interrompu. Analogue au paramètre *h*, le paramètre *p* permet de définir un seuil applicable au rapport entre deux mesures concernant tous les processus : le nombre de *pages déjà référencées* (défini à la section relative à l'algorithme de repositionnement de page) et le nombre de défauts de page cumulés par le processus durant la dernière seconde. Un rapport élevé indique que le processus s'emballe. Un processus est candidat à la suspension (soit parce qu'il s'emballe soit parce qu'il contribue à l'emballement de l'ensemble) si :

$$\frac{\text{nombre de pages référencées au cours de la dernière seconde}}{\text{nombre de défauts de page au cours de la dernière seconde}} > \frac{1}{p}$$

Par défaut, *p* prend la valeur 4, ce qui signifie qu'un processus est considéré comme s'emballant (et donc candidat à la suspension) dès lors que le rapport entre les pages référencées et les défauts de page cumulés au cours de la dernière seconde est supérieur à 25 %. Une valeur faible de *p* (pouvant atteindre zéro, le test n'effectuant pas réellement la division) autorise le système à s'approcher davantage des conditions d'emballement avant que ne se déclenche l'interruption des processus. La valeur zéro signifie qu'aucun processus ne peut être interrompu par le système de contrôle de charge mémoire.

Paramètre m

Le paramètre *m* détermine la limite inférieure du degré de multiprogrammation. Le degré de multiprogrammation est défini comme le nombre de processus actifs (non suspendus). (Chaque processus compte pour un, quel que soit le nombre de routines qu'il exécute.) Ne sont pas comptabilisés le processus noyau et les processus à priorité fixe inférieure à 60 (1), à mémoire fixe (2) ainsi que les événements en attente (3), aucun processus de cette catégorie ne pouvant être interrompu. La valeur 2 par défaut garantit qu'au moins deux processus utilisateur sont toujours activables. Une valeur inférieure de*m*, lorsqu'elle est autorisée, indique que par moments un seul processus utilisateur peut être actif. Des valeurs supérieures de *m* retirent au système de contrôle de charge la possibilité d'interrompre des processus. Ce paramètre dépend fortement de la configuration et de la charge. Une valeur trop faible de *m* dans une configuration étendue déclenche trop aisément la suspension des processus, une valeur trop élevée de *m* dans une configuration réduite ne confère pas au système de contrôle une sensibilité suffisante. La valeur 2 (par défaut) est le meilleur compromis pour une configuration réduite mais risque d'être trop faible pour une configuration plus étendue, pouvant et devant prendre en charge des dizaines de processus pour exploiter les ressources disponibles.

Par exemple, si une configuration et une charge données peuvent gérer l'exécution simultanée d'environ 25 processus, mais qu'au-delà, il y a risque d'emballement, il est conseillé de fixer à 25 la valeur de *m*.

Paramètre e

Chaque fois qu'un processus interrompu est réactivé, il est exempt de toute interruption pendant une période de *e* secondes. Ce laps de temps est ménagé pour assurer que le coût élevé (en E/S disque) que représente le chargement des pages d'un processus interrompu est compensé par une progression raisonnable des processus. La valeur par défaut de *e* est 2 secondes.

Une fois par seconde, le programmateur (processus 0) examine toutes les grandeurs citées ci-dessus, collectées pendant l'intervalle d'une seconde précédent. Cet examen lui permet de déterminer si les processus doivent être suspendus ou activés. Tous les processus désignés aptes à être suspendus par les paramètres *p* et *e* sont marqués comme tels.

Lorsqu'un de ces processus reçoit la CPU en mode utilisateur, il est interrompu (sauf si la suspension réduit à moins de *m* le nombre de processus actifs). Le critère de mode utilisateur est appliqué pour empêcher qu'un processus soit interrompu alors que des opérations système critiques sont exécutées sur son compte. Si les conditions d'emballement perdurent au cours des intervalles d'une seconde suivants, les autres processus répondant aux critères de *p* et *e* sont marqués pour être interrompus. Lorsque le programmateur détermine que les conditions d'intervalle de sécurité sont remplies, chaque seconde, un certain nombre de processus interrompus sont remis dans la file d'exécution (réactivés).

Les processus interrompus sont réactivés : (1) par ordre de priorité, (2) par ordre d'interruption. Ils ne sont pas tous activés en même temps. Leur nombre est calculé selon une formule qui établit le nombre de processus actifs à ce moment-là et en réactive 1/5ème ou augmente selon une progression régulière croissante le nombre minimum (c'est l'option la plus élevée qui est choisie). Par cette stratégie prudente, l'accroissement du degré de multiprogrammation est réalisé par tranche d'environ 20 % par seconde. Ainsi, la réactivation s'effectue à un rythme relativement lent pendant la première seconde qui suit la période de sécurité, puis à un rythme plus soutenu au cours des secondes suivantes. Si la surcharge mémoire se reproduit pendant la réactivation, l'opération s'interrompt et les processus marqués pour la réactivation sont de nouveau marqués comme étant interrompus. Les autres processus sont interrompus conformément aux règles précédentes.

Les six paramètres de l'utilitaire de contrôle de charge mémoire peuvent être définis par l'administrateur système via la commande **schedtune**. Les techniques d'optimisation de l'utilitaire sont décrites au chapitre 7 : "Contrôle et optimisation de la mémoire".

Affectation et réclamation d'emplacements dans l'espace de pagination

AIX prend en charge deux schémas d'affectation des emplacements dans l'espace de pagination. Avec l'algorithme normal, de *post-affectation*, un emplacement de pagination n'est affecté à une page de mémoire virtuelle que lorsque cette page a été lue ou qu'une écriture y a été effectuée. C'est la première fois que le contenu de la page a un rôle pour le programme en cours.

Un grand nombre de programmes exploite ce type d'affectation en attribuant des intervalles d'adresses de mémoire virtuelle aux plus grandes structures, celles-ci étant par la suite utilisées en fonction des besoins. Les pages des intervalles d'adresses de mémoire virtuelle qui ne sont jamais sollicitées ne requièrent ni trames de mémoire réelle ni emplacements dans l'espace de pagination.

Cette technique n'est cependant pas sans risque. Si tous les programmes exécutés sur une machine se trouvent simultanément dans une situation de taille maximale, l'espace de pagination peut être saturé. Certains programmes riquent de ne pas être menés à terme.

Le second schéma utilisé par AIX a été spécialement conçu pour ce type de situation ou pour les installations sur lesquelles le non-aboutissement d'un programme représente un coût prohibitif. Cet algorithme dit d'*affectation anticipée* permet d'affecter le nombre approprié d'emplacements dans l'espace de pagination en même temps que l'intervalle d'adresse de mémoire virtuelle, par exemple à l'aide de **malloc**. Si le nombre d'emplacements est insuffisant pour le **malloc**, un code d'erreur est généré. La ligne d'appel de cet algorithme est :

```
export PSALLOC=early
```

Dès lors, tous les programmes **exec** dans l'environnement utiliseront l'algorithme d'affectation anticipée. Cette ligne n'a aucune incidence sur le shell courant.

L'affectation anticipée est particulièrement intéressante pour l'analyse des performances, du fait de ses implications sur la taille de l'espace de pagination. Nombreux sont les programmes qui utilisent fréquemment **malloc** pour une utilisation de l'espace "à la carte". Activer l'algorithme d'affectation anticipée pour ces programmes entraîne une multiplication des besoins en espace de pagination. S'il est normalement conseillé d'utiliser un espace de pagination de taille double de celle de la mémoire réelle du système, dans le cas de

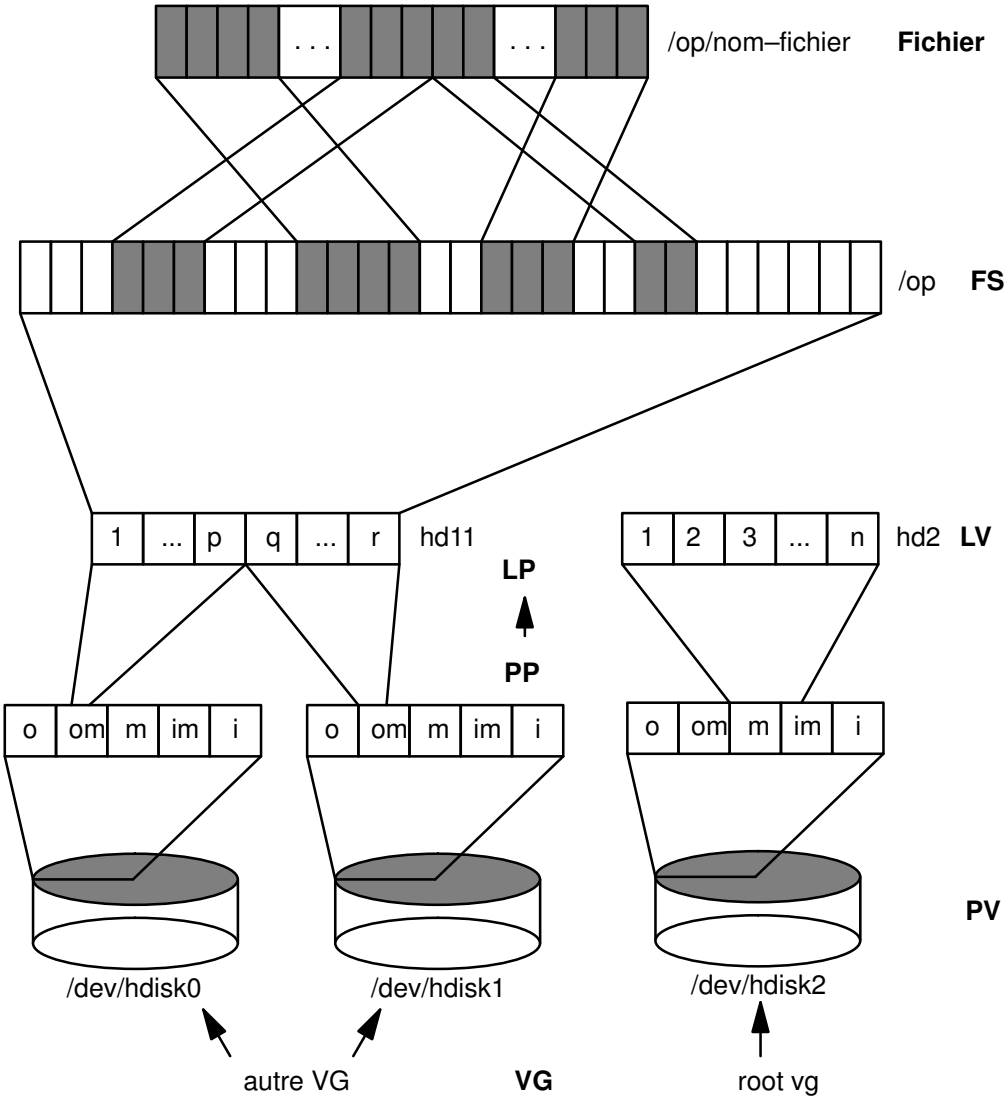
systèmes utilisant PSALLOC=early, la taille préconisée doit être au moins quadruple de celle de la mémoire réelle. Et ceci n'est qu'une base de départ. Il vous faut en fait analyser les besoins en mémoire virtuelle de la charge de travail et affecter les espaces de pagination correspondants. Par exemple, le serveur AIX windows peut requérir, avec l'affectation anticipée, 250 Mo d'espace de pagination.

Rappelons également que ces emplacements ne sont libérés qu'à la fin du processus (et non des routines) ou par l'appel système **disclaim**. Ils ne peuvent *pas* être libérés via **free**.

Pour en savoir plus sur le contrôle et l'affectation de l'espace de pagination, reportez-vous à "Position et taille des espaces de pagination", page 4-26.

Gestion AIX du stockage sur disque fixe

La figure ”Organisation des données sur disque fixe (sans option miroir)” illustre la hiérarchie des structures utilisées par AIX pour gérer le stockage sur disque fixe. Chaque unité de disque, appelée *volume physique* (PV), porte un nom du type `/dev/hdisk0`. Si le volume est en cours d'utilisation, il appartient à un *groupe de volumes* (VG). Tous les volumes physiques membres d'un groupe de volumes sont divisés en *partitions physiques* (PP) de même taille (par défaut, 2 Mo pour des groupes comportant des volumes physiques inférieurs à 300 Mo et 4 Mo dans les autres cas). Pour faciliter l'affectation d'espace, chaque volume physique est divisé en cinq régions (outer_edge, outer_middle, center, inner_middle et inner_edge). Le nombre de partitions physiques implantées dans chaque région varie en fonction de la capacité totale de l'unité de disque.



Organisation des données sur disque fixe (sans option miroir)

Dans chaque groupe de volumes sont définis un ou plusieurs *volumes logiques* (LV). Chaque volume logique est constitué d'une ou de plusieurs partitions logiques. Chaque partition logique correspond à au moins une partition physique. Si le volume logique est mis en miroir, des partitions physiques supplémentaires sont affectées pour stocker les copies supplémentaires de chaque partition logique. Les partitions logiques sont numérotées séquentiellement, mais les partitions physiques sous-jacentes ne sont pas nécessairement consécutives ou contiguës.

Les volumes logiques peuvent assurer certaines fonctions système, telle la pagination. S'ils renferment des données système ou utilisateur ou des programmes, ils disposent chacun d'un système de fichiers journalisés (JFS) unique. Ce dernier est composé d'un pool de blocs de la taille d'une page (4 096 octets). Lorsque des données sont inscrites dans un fichier, un ou plusieurs blocs lui sont affectés. Les blocs peuvent être contigus ou non (et contigus ou non à ceux déjà affectés au fichier).

Sous AIX version 4.1, un système de fichiers donné peut être défini avec une taille de fragment inférieure à 4 096 octets. La taille de fragment est de 512, 1 024 ou 2 048 octets. Ce qui permet de stocker plus efficacement les fichiers peu volumineux.

La figure "Organisation des données sur disque fixe (sans option miroir)" illustre ce qui peut se produire lorsqu'un système de fichiers n'a pas été réorganisé depuis longtemps. Le fichier /**op**/**filename** est réparti sur un grand nombre de blocs physiquement distants les uns des autres : la lecture séquentielle de ce fichier exige plusieurs opérations de recherche, coûteuses en temps.

Au niveau conceptuel, un fichier AIX est une chaîne contiguë et séquentielle d'octets, mais la réalité physique est parfois bien différente : *le fichier peut avoir été fragmenté* à la suite de multiples extensions des volumes logiques ou d'opérations d'affectation/ libération/ réaffectation au sein du système de fichiers. Un système de fichiers est dit fragmenté lorsque son espace disponible est composé de multiples parcelles qui rendent impossible l'écriture d'un nouveau fichier sur des blocs contigus.

L'accès aux fichiers implantés sur un système de fichiers très fragmenté exige de multiples opérations de recherche et induit des temps de réponse en E/S bien supérieurs, du fait des délais de recherche. Par exemple, l'accès séquentiel demande plus d'opérations de recherche si le fichier est dispersé sur des parcelles physiquement éloignées que s'il est concentré sur une ou quelques parcelles contiguës. Il en va de même pour l'accès aléatoire à un fichier.

L'impact de la dispersion d'un fichier sur les performances d'E/S est moindre si le fichier est en mémoire tampon. Lorsqu'un fichier est ouvert sous AIX, il est mappé à un segment de donnée persistant en mémoire virtuelle. Ce segment joue le rôle de tampon virtuel, les blocs du fichier renvoient directement aux pages du segment. VMM gère les pages de segment en lisant à la demande les blocs du fichier qui y sont contenus (au fur et à mesure de leur ouverture). Il arrive que VMM réécrive une page sur le bloc correspondant du fichier sur disque ; mais, en général, il conserve en mémoire les pages récemment appelées. Ces pages restent donc plus longtemps en mémoire et l'accès des fichiers logiques aux blocs correspondants s'effectue sans passer par le disque physique.

A un certain stade, l'utilisateur ou l'administrateur système peut décider de réorganiser la position des fichiers dans les volumes logiques et la position des volumes logiques dans les volumes physiques, de façon à réduire la fragmentation et à répartir plus uniformément la charge totale des E/S. Pour en savoir plus sur la détection et la correction des problèmes liés à la répartition sur disque et à la fragmentation, reportez–vous à "Contrôle et optimisation des E/S disque", page 8-1.

Lecture séquentielle anticipée

Le gestionnaire VMM tente d'anticiper les besoins en pages d'un fichier séquentiel en observant le mécanisme d'accès des programmes à ce fichier. Si le programme accède à deux pages successives, VMM en déduit que l'accès au fichier est séquentiel et anticipe les lectures suivantes. Ces lectures s'effectuent parallèlement à l'exécution du programme, de sorte que celui-ci dispose des données plus rapidement, VMM n'ayant pas à attendre qu'il accède à la page suivante pour lancer une opération d'E/S. Le nombre de pages lues par anticipation dépend de deux seuils VMM :

minpgahead Nombre de pages lues par anticipation dès que VMM suppose que l'accès est séquentiel. Si le programme continue d'accéder au fichier séquentiellement, la lecture suivante portera sur le double de pages défini à **minpgahead**, puis la suivante sur quatre fois le nombre de pages défini à **minpgahead**, etc., jusqu'à atteindre le nombre de pages défini à **maxpgahead**.

maxpgahead Nombre maximum de pages du fichier séquentiel que VMM lit par anticipation.

Si le programme s'écarte du mécanisme d'accès séquentiel et accède à une page du fichier dans un autre ordre, la lecture anticipée s'interrompt. Elle reprend, avec le nombre de pages défini à **minpgahead**, si VMM constate que le programme revient à un accès séquentiel. Les valeurs de **minpgahead** et **maxpgahead** sont définies via la commande **vmtune**. Pour en savoir plus, reportez-vous à "Optimisation des lectures séquentielles anticipées", page 8-11.

Ecriture différée

Pour accroître les performances d'écriture, limiter le nombre en mémoire de pages de fichier modifiées, réduire la charge du système et minimiser la fragmentation du disque, le système de fichiers divise chaque fichier en partitions de 16 ko. Les pages d'une partition donnée ne sont inscrites sur le disque que lorsque le programme a écrit le premier octet de la partition suivante. Ce n'est qu'à ce stade que le système de fichiers force l'écriture des quatre pages modifiées de la première partition sur le disque. Les pages de données restent en mémoire jusqu'à ce que leur trame soit réutilisée, stade à partir duquel aucune autre opération d'E/S n'est requise. Si un programme accède à l'une des pages avant que les trames ne soient réutilisées, aucune E/S n'est nécessaire.

S'il reste beaucoup de page modifiées non réutilisées en mémoire, le démon **sync** les écrit sur le disque, ce qui peut entraîner une utilisation anormale du disque. Pour répartir plus efficacement les activités d'E/S par rapport à la charge de travail, vous pouvez activer l'écriture différée aléatoire pour indiquer au système le nombre de pages à conserver en mémoire avant de les écrire sur disque. Le seuil d'écriture différée aléatoire est défini sur la base des fichiers. Cette action entraîne l'écriture des pages sur disque avant l'exécution du démon **sync** ; les E/S sont ainsi réparties plus uniformément.

Vous pouvez modifier la taille des partitions d'écriture différées et le seuil d'écriture différée aléatoire via la commande **vmtune**.

Fichiers mappés et écriture différée

Les fichiers AIX normaux sont automatiquement mappés à des segments. Ainsi, l'accès normal à un fichier ne nécessite plus de transiter par les tampons du noyau et les routines d'E/S de bloc, ce qui permet aux fichiers d'utiliser davantage de mémoire lorsqu'un supplément de mémoire est disponible (la mise en mémoire cache des fichiers n'est pas limitée à la zone tampon déclarée du noyau).

Les fichiers peuvent être mappés explicitement par la commande **shmat** ou **mmap**, mais cette opération ne fournit pas d'espace mémoire supplémentaire pour la mise en mémoire cache. Les applications qui utilisent **shmat** ou **mmap** et accèdent au fichier mappé par une adresse au lieu de recourir à **read** et **write** évitent les longueurs de parcours inhérentes aux opérations d'appel système, mais perdent le bénéfice de la fonction d'écriture différée. Lorsque les applications n'utilisent pas la sous-routine **write**, les pages modifiées ont tendance à s'accumuler en mémoire et sont écrites de façon aléatoire quand l'algorithme de repositionnement de page VMM ou le démon sync les purge. Dans ces conditions, l'écriture sur disque se fait par petits bouts, au détriment de toute efficacité d'exploitation du disque et de la CPU, et la fragmentation qui en découle risque de ralentir les lectures ultérieures du fichier.

Régulation des E/S

Les utilisateurs des versions AIX antérieures à la version 3.2 subissaient parfois des temps de réponse très longs sur des applications interactives, lorsqu'une autre application exécutait une importante opération d'écriture sur le disque. La plupart des écritures étant asynchrones, il peut se créer des files d'attente d'E/S FIFO de plusieurs mégaoctets, dont l'exécution demande plusieurs secondes. Si chaque lecture sur disque monopolise plusieurs secondes pour parcourir les files d'attente, les performances d'un processus interactif sont sérieusement amoindries. Pour résoudre ce problème, VMM propose une option de contrôle des écritures appelée *régulation des E/S.*

La régulation des E/S ne modifie ni l'interface ni la logique de traitement des E/S. Elle se borne à limiter le nombre des E/S en attente sur un fichier. Lorsqu'un processus tente de dépasser cette limite, il est interrompu jusqu'à ce qu'un nombre sufffisant de requêtes en attente ait été traité pour atteindre le seuil minimal. Pour en savoir plus, reportez-vous à "Régulation des E/S disque", page 8-13.

Pile de disques

Une pile de disques est un ensemble de disques gérés comme un tout. Plusieurs algorithmes de gestion engendrent divers degrés de performances et/ou d'intégrité des données. Ces algorithmes de gestion sont identifiés par différents niveaux RAID. (RAID signifie "redundant array of independent disks".) Les niveaux RAID sous les versions 3.2.5 et 4 définis dans l'architecture sont les suivants :

RAID0 Les données sont inscrites sur des unités physiques consécutives, avec un nombre fixe de blocs de 512 octets par écriture. Cette technique est comparable à la technique de répartition en bandes des données sur disque : elle offre les mêmes caractéristiques au niveau de l'intégrité des données que des unités de disque indépendantes classiques. Autrement dit, l'integrité des données dépend entièrement de la fréquence et de la validité des sauvegardes. Ce niveau est analogue à la fonction de répartition en bande décrite à "Performances et répartition des volumes logiques", page 8-15.

RAID1 Les données sont réparties en bandes sur les unités, comme pour le niveau RAID0, mais la moitié des unités sont utilisées comme unités miroir. RAID1 répond à certains problèmes d'intégrité et de disponibilité pouvant survenir avec RAID0 en cas de défaillance d'une unité unique, mais n'offre pas plus de solutions que RAID0 en cas de défaillance d'une ou plusieurs unités. Des sauvegardes consciencieuses sont indispensables. Ce niveau est analogue à la fonction miroir des volumes logiques proposée par le gestionnaire de volumes logiques.

RAID3 Les données sont réparties en bandes octet par octet sur un ensemble d'unités de données, une unité de parité distincte contenant un octet de parité pour chaque position d'octet dans les unités de données. En cas de défaillance d'une unité unique, son contenu est reconstitué sur la base de l'octet de parité et des octets de données restants. Dans cette technique, l'unité de parité devient un goulot d'étranglement en terme de performances : une écriture doit y être effectuée à chaque écriture sur *un* des autres disques.

RAID5 Les données sont réparties bloc par bloc (de 512 octets), mais des portions d'un certain nombre d'unités (pas forcément toutes) sont réservées aux informations de parité. Ce qui alourdit encore l'écriture des informations de parité.

Les unités RAID doivent être considérées comme une solution en terme d'intégrité et de disponibilité, mais non de performances. Les configurations RAID étendues sont généralement limitées par le fait que chaque RAID est raccordé à une carte SCSI unique. Du point de vue des performances, la prise en charge d'un nombre donné d'unités de disque est plus efficace avec des unités RAID raccordées à plusieurs cartes SCSI qu'avec une seule unité RAID de grande taille.

Chapitre 3. Introduction au multitraitement

”Répartir la charge pour alléger le travail”, tel est le postulat qui a conduit au développement des systèmes multiprocesseur. A un moment donné, la vitesse de traitement d’un processeur se heurte à une limite technologique. Si un seul processeur ne parvient pas à gérer correctement la charge du système, une solution possible est de multiplier les processeurs.

Le succès de cette solution dépend bien entendu de l’ingéniosité de ses concepteurs, et encore faut-il que la charge de travail s’y prête : s’il est évident que multiplier les employés répondant à un numéro vert accroît l’efficacité du service, multiplier les conducteurs d’une automobile ne présente aucun intérêt.

Pour que la migration d’un système monoprocesseur en système multiprocesseur se traduise par l’amélioration de ses performances, les conditions suivantes doivent être réunies :

- La charge de travail est limitée par la vitesse de traitement et sature l’unique processeur du système.
- La charge de travail contient plusieurs éléments gourmands en temps processeur (transactions, calculs complexes, etc.) qui peuvent être traités simultanément et de manière indépendante.
- Le processeur existant ne peut pas être mis à niveau et aucun processeur seul n’offre la vitesse de traitement nécessaire.
- Plusieurs éléments (base de données centralisée, etc.) incitent à répartir la charge de travail entre plusieurs systèmes monoprocesseur.

D’une manière générale, il est préférable d’opter pour une solution mettant en oeuvre un seul processeur lorsque cela est possible car l’association de plusieurs processeurs pose des problèmes de performances, négligeables, voire inexistants, dans les systèmes monoprocesseur. En particulier, si la condition 2 n’est pas remplie, les performances d’un système multiprocesseur peuvent être moindres que celles d’un système monoprocesseur.

Bien que les applications unifilaires inchangées fonctionnent en général correctement dans un environnement multiprocesseur, leurs performances s’en trouvent souvent modifiées de manière imprévue. La mise en place d’un système multiprocesseur peut améliorer le débit d’un système et parfois raccourcir le temps d’exécution d’applications multifiles complexes, mais il réduit rarement le temps de réponse de chaque commande unifilaire.

Pour optimiser les performances d’un système multiprocesseur, il convient de connaître la dynamique du système d’exploitation et du fonctionnement du matériel, spécifique des environnements multiprocesseurs.

Multitraitement symétrique (SMP) – concepts et architecture

Comme toute modification qui accroît la complexité d'un système, l'utilisation de plusieurs processeurs nécessite de modifier la conception pour assurer un fonctionnement et des performances satisfaisants. Du fait de sa plus grande complexité, le système multiprocesseur nécessite davantage d'échanges matériel/logiciel et davantage de coordination dans la conception matérielle et logicielle qu'un système à processeur unique. Les diverses combinaisons de conception et d'échanges augmentent d'autant le nombre d'architectures possibles.

Ce chapitre décrit les principaux problèmes posés par les systèmes multiprocesseurs et les solutions proposées par le système AIX et le ESCALA.

Lors de la conception d'un système multiprocesseur, la décision la plus importante est sans doute le choix entre un système symétrique ou asymétrique.

Les aspects importants de la conception sont les suivants :

- Multiprocesseurs symétriques et asymétriques
- Sérialisation des données
- Granularité du verrouillage
- Charge liée au verrouillage
- Cohérence des mémoires cache
- Affinité avec un processeur
- Conflits d'utilisation de mémoire et de bus

Multiprocesseurs symétriques et asymétriques

Dans un système multiprocesseur asymétrique, chaque processeur joue un rôle particulier : gestion des entrées-sorties, exécution de programmes, etc. Ce type de système présente des avantages et des inconvénients :

- L'affectation de tâches particulières à un seul processeur permet de limiter, voire de supprimer, les incidents liés à la sérialisation des données et à la cohérence des mémoires cache (voir plus loin). Elle permet également à certains éléments logiciels de fonctionner comme dans un système monoprocesseur.
- Dans certains cas, le traitement des opérations d'entrées-sorties et des programmes est accéléré car il n'est pas en concurrence avec d'autres éléments du système d'exploitation ou de la charge de travail pour l'accès au processeur.
- Dans d'autres cas, le traitement des opérations d'entrées-sorties et des programmes est ralenti car les processeurs ne sont pas tous disponibles pour gérer les pointes de charge.
- L'affectation de tâches particulières à chaque processeur permet de limiter les conséquences d'un incident à une partie restreinte du système.

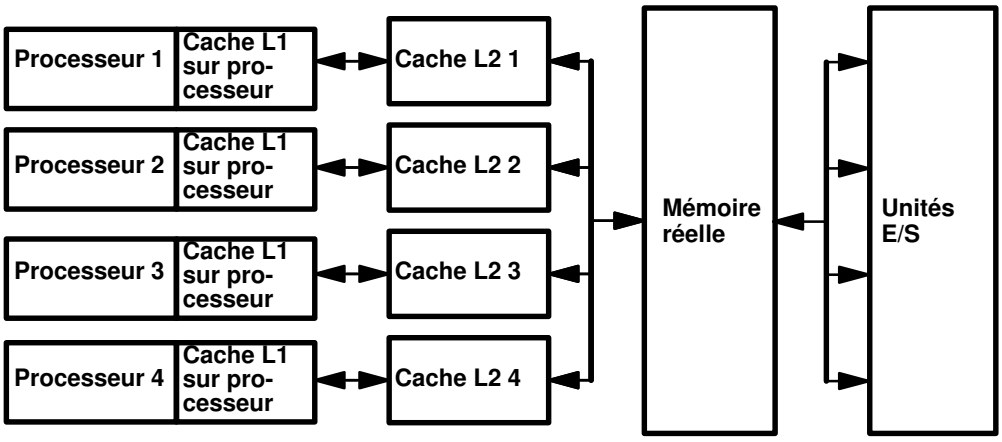
Dans un système multiprocesseur symétrique, tous les processeurs sont pratiquement identiques et exécutent des fonctions similaires :

- Ils disposent tous d'espaces d'adresses virtuelles et réelles identiques.
- Ils peuvent tous exécuter toutes les routines du système.
- Ils peuvent tous gérer une interruption externe. (Les interruptions internes sont gérées par le processeur qui exécute le flot d'instructions qui les a générées.)
- Tous les processeurs peuvent lancer une opération d'entrée-sortie.

Du fait de cette interchangeabilité, n'importe quel processeur peut exécuter n'importe quelle fonction. C'est principalement aux concepteurs de matériel et de logiciels qu'il incombe de

garantir cette flexibilité. Un système symétrique fait cependant apparaître les limites du multitraitement d'une charge de travail.

La famille ESCALA ne contient que des systèmes multiprocesseur symétriques, dont la figure "Système multiprocesseur symétrique" donne un exemple. AIX version 4.1 ne prend en charge que ce type de système. D'autres types de système peuvent présenter une configuration différente de leur mémoire cache.



Système multiprocesseur symétrique

Bien que les systèmes multiprocesseur ESCALA soient techniquement symétriques, les logiciels y introduisent un peu d'asymétrie. Au départ, pendant l'amorçage, un seul processeur a le contrôle. Le premier processeur démarré est appelé "processeur maître". Afin que les logiciels écrits par l'utilisateur continuent de fonctionner correctement lors de la transformation de l'environnement monoprocesseur en environnement multiprocesseur, les pilotes d'unité et les extensions de noyau qui ne sont pas explicitement conçus pour un système multiprocesseur sont contraints de s'exécuter uniquement sur le processeur maître. Cette contrainte s'appelle "canalisation".

Sérialisation des données

Tout élément de mémoire qui peut être lu ou écrit par plusieurs routines peut être modifié pendant l'exécution du programme. Ce phénomène se produit généralement dans les environnements de multiprogrammation et les environnements multitraitement. L'apparition de systèmes multiprocesseur accroît la portée et l'importance de ce phénomène :

- La prise en charge de plusieurs processeurs et des routines incitent à la création d'applications partageant les données entre plusieurs routines.
- Le noyau ne peut plus résoudre les incidents liés à la sérialisation des données en désactivant simplement les interruptions.

Pour éviter une catastrophe, les programmes qui partagent des données doivent y accéder de manière *séquentielle* et non plus en parallèle. Avant d'accéder à une donnée partagée, chaque programme doit vérifier qu'aucun autre programme (y compris une copie de lui-même s'exécutant au sein d'une autre routine) ne modifie la donnée.

Le principal mécanisme utilisé pour éviter les interférences entre programmes est le *verrouillage*. Un verrou est une abstraction qui représente l'autorisation d'accès à un ou plusieurs éléments de données. Les demandes de verrouillage ou de déverrouillage sont atomiques : elles s'effectuent de telle sorte que ni les interruptions, ni l'accès multiprocesseur n'affectent le résultat. Pour accéder à une donnée partagée, un programme doit obtenir son déverrouillage. Si un autre programme (ou une autre routine exécutant le même programme) a déjà obtenu ce déverrouillage et l'accès à la donnée, le programme demandeur doit attendre que l'accès lui soit accordé.

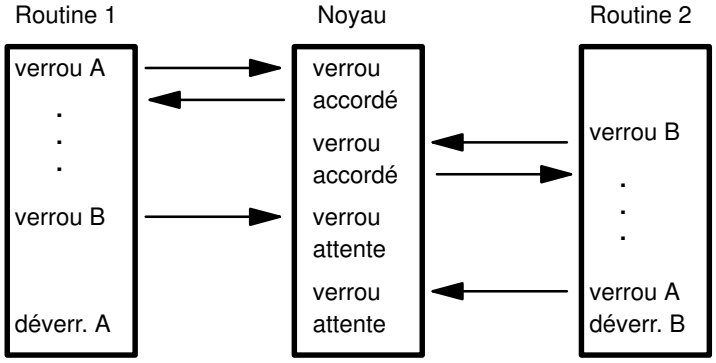
Outre les temps d'attente d'accès aux données, la sérialisation accroît le nombre de périodes pendant lesquelles une routine n'est pas diffusable. Pendant ces périodes, les

autres routines sont susceptibles de modifier les lignes de la mémoire cache de cette routine, ce qui augmente le délai de latence de la mémoire lorsque la routine obtient finalement le déverrouillage de la donnée et redevient diffusable.

Le noyau AIX contient un grand nombre de données partagées et doit donc procéder à la sérialisation des données en interne. Cela peut provoquer des retards de sérialisation, y compris dans les applications qui ne partagent pas de données avec d'autres programmes, car les fonctions du noyau utilisées par l'application doivent sérialiser les données partagées du noyau.

Granularité du verrouillage

Un programmeur qui travaille dans un environnement multiprocesseur doit décider du nombre de verrous distincts à créer pour les données partagées. Si un seul verrou doit sérialiser l'ensemble des données partagées, le risque de conflit d'accès est relativement élevé. Si chaque donnée dispose d'un verrou, le risque de conflit est relativement faible. Cependant, toute demande supplémentaire de verrouillage ou de déverrouillage utilise du temps processeur et l'existence de plusieurs verrous peut provoquer un blocage. La figure "Blocage" présente le cas le plus simple de blocage : la routine 1 a obtenu le déverrouillage de la donnée A et attend le déverrouillage de la donnée B et la routine 2 a obtenu le déverrouillage de la donnée B et attend le déverrouillage de la donnée A. Aucun des deux programmes ne peut obtenir le **déverrouillage** qui mettrait fin au blocage. Pour éviter les blocages, on établit généralement un protocole par lequel tous les programmes qui utilisent un ensemble de verrous doivent toujours obtenir le déverrouillage dans le même ordre.



Blocage

Charge liée au verrouillage

Les demandes, délais et libérations de verrous augmentent la charge du système :

- Un programme qui prend en charge le multitraitement effectue en permanence les mêmes opérations de verrouillage et de déverrouillage, même s'il s'exécute dans un environnement monoprocesseur ou qu'il est le seul à utiliser un verrou donné dans un système multiprocesseur.
- Si une routine demande le déverrouillage d'une donnée déjà obtenu par une autre routine, il se peut que la routine demandeuse tourne un moment, soit mise en veille et, si c'est possible, qu'une autre routine soit diffusée. Cela consomme du temps système.
- La présence de verrous très utilisés fixe une limite maximale au débit du système. Par exemple, si un programme donné consacre 20 % de son temps d'exécution à l'utilisation d'un verrou à exclusion mutuelle, 5 exemplaires au plus de ce programme peuvent s'exécuter simultanément, *quel que soit le nombre de processeurs* du système. En réalité, même 5 exemplaires seulement d'un programme ne seront jamais suffisamment synchronisés pour éviter les temps d'attente ("Échelle de débit dans un système multiprocesseur", page 3-8).

Cohérence des mémoires cache

Les concepteurs de systèmes multiprocesseur veillent tout particulièrement à assurer la cohérence des mémoires cache. Mais cette cohérence ne peut être garantie qu'au détriment des performances. Pour comprendre pourquoi, il convient de connaître les difficultés rencontrées :

Si chaque processeur possède une mémoire cache (voir figure "Système multiprocesseur symétrique", page 3-3), qui indique l'état des différentes parties de la mémoire, il se peut que plusieurs mémoires cache contiennent des copies d'une même ligne. Il se peut également qu'une ligne contienne plusieurs données dotées d'un verrou. Si deux routines apportent des modifications à ces données dans l'ordre correct, deux versions différentes et incorrectes de la ligne de mémoire risquent d'être enregistrées dans les mémoires cache. Le système n'est plus cohérent car il contient deux versions différentes du contenu d'une zone de mémoire spécifique.

Pour rétablir la cohérence des mémoires cache, on conserve généralement une des lignes et on invalide les autres. Bien que l'invalidation soit effectuée par le matériel, sans intervention logicielle, tout processeur dont une ligne de mémoire cache a été invalidée indiquera une absence en mémoire cache, avec le délai correspondant, la prochaine fois qu'il recevra une demande concernant cette ligne.

Pour une étude détaillée de l'architecture d'adressage et des opérations sur la mémoire cache ESCALA, reportez-vous à l'annexe C, "Mémoire cache et adressage".

Affinité avec un processeur

Si une routine est interrompue, puis rediffusée sur le même processeur, il se peut que la mémoire cache de ce processeur contienne encore des lignes appartenant à cette routine. Si la routine est diffusée sur un autre processeur, elle rencontrera probablement une série d'absences en mémoire cache avant que la partie active de sa mémoire cache soit récupérée de la RAM. Cependant, si une routine diffusable doit attendre que le processeur sur lequel elle s'exécutait auparavant soit de nouveau disponible, elle risque d'attendre encore plus longtemps.

L'affinité avec un processeur est la diffusion d'une routine sur le processeur sur lequel elle s'exécutait auparavant. Le degré d'affinité avec un processeur doit varier proportionnellement à la taille de la partie active de la mémoire cache et de manière inversement proportionnelle au temps écoulé depuis la dernière diffusion.

Sous AIX version 4.1, l'affinité avec un processeur peut être effectuée via une *liaison* entre une routine et un processeur. Une routine liée à un processeur ne peut s'exécuter que sur ce processeur, quel que soit l'état des autres processeurs du système.

Conflits d'utilisation de mémoire et de bus

Dans un système monoprocesseur, les conflits d'utilisation de ressources internes (blocs de mémoire, bus de mémoire ou d'entrées-sorties, etc.) sont généralement insignifiants en terme de temps système. Dans un système multiprocesseur, ils peuvent prendre davantage d'importance, notamment si les algorithmes de cohérence des mémoires cache augmentent le nombre d'accès à la mémoire vive.

Performances des systèmes SMS

Répartition de la charge de travail

Le premier problème de performances spécifique des systèmes SMS est la *répartition de la charge de travail*, c'est-à-dire, l'exploitation efficace des *n* processeurs disponibles. En effet, si seul un processeur d'un système quadriprocesseur est actif à un instant donné, le système n'est pas plus performant qu'un système monoprocesseur. Il risque même de l'être moins, du fait de l'existence du code destiné à éviter les interférences entre processeurs.

La répartition de la charge de travail est le pendant de la sérialisation des données. Si les logiciels de base ou la charge de travail, ou leur interaction, requièrent la sérialisation des données, la répartition de la charge de travail en pâtit.

Mieux vaut une répartition de la charge de travail moins équilibrée du fait d'une plus grande affinité avec un processeur. Celle-ci améliorant l'efficacité de la mémoire cache, l'exécution du programme s'en trouve accélérée. La répartition de la charge de travail est moindre (sauf s'il y a plus de routines diffusables disponibles), mais le temps de réponse est réduit.

La *répartition des processus*, lesquels font partie de la charge de travail, est la proportion de routines diffusables en permanence sur un processus multiroutines.

Débit

Le débit d'un système SMS dépend principalement des éléments suivants :

- Une répartition *constamment* très bonne de la charge de travail. Davantage de routines diffusables que de processeurs à certains instants ne compense pas l'inactivité des processeurs à d'autres.
- Le nombre de conflits d'accès.
- Le degré d'affinité avec un processeur.

Temps de réponse

Le temps de réponse d'un programme dans un système SMS est fonction des éléments suivants :

- Le niveau de répartition des processus du programme. Si celui-ci a en permanence plusieurs routines diffusables, son temps de réponse s'améliorera probablement dans un environnement SMS. S'il est constitué d'une seule routine, son temps de réponse sera au mieux comparable à celui d'un système monoprocesseur équivalent.
- Le nombre de conflits d'accès entre plusieurs instances d'un programme ou entre plusieurs programmes différents utilisant les mêmes verrous.
- Le degré d'affinité avec un processeur. Si chaque diffusion d'un programme a lieu vers des processeurs différents qui ne disposent pas des lignes de mémoire cache du programme, celui-ci risque de s'exécuter plus lentement que dans un système monoprocesseur comparable.

Adaptation des programmes à un environnement SMS

Les termes qui suivent permettent d'indiquer dans quelle mesure un programme a été modifié ou créé pour fonctionner dans un environnement SMS :

Adaptation minimale au SMS	Suppression dans un programme de toute action (accès non sérialisé aux données partagées, par exemple) susceptible de provoquer des incidents dans un environnement SMS. Utilisé seul, ce terme désigne généralement un programme qui a subi les modifications minimales pour fonctionner correctement dans un système SMS.
Adaptation correcte au SMS	Suppression dans un programme de toute action susceptible de provoquer des incidents liés à son fonctionnement ou à ses performances dans un système multiprocesseur symétrique. Un programme qui a subi une adaptation correcte a généralement subi également une adaptation minimale. Il a subi des modifications supplémentaires pour réduire au minimum les goulets d'étranglement en formation.
Adaptation totale au SMS	Ajout dans un programme de fonctions destinées à permettre son fonctionnement correct dans un système SMS. Un programme qui a subi cette adaptation a généralement subi aussi les deux autres adaptations.

Charge de travail d'un SMS

L'ajout de processeurs supplémentaires affecte les performances principalement en fonction des caractéristiques de la charge de travail. Les sections suivantes décrivent ces caractéristiques et leurs effets sur les performances.

- "Multitraitement de la charge de travail"
- "Échelle de débit dans un système multiprocesseur"
- "Temps de réponse d'un système multiprocesseur"

Multitraitement de la charge de travail

Les systèmes d'exploitation à multiprogrammation comme AIX qui traitent d'énormes charges de travail sur des ordinateurs rapides tels que le ESCALA donnent l'impression que plusieurs opérations ont lieu simultanément. En réalité, nombre de charges de travail ne contiennent jamais un grand nombre de routines diffusables, même lorsqu'elles sont traitées sur un système monoprocesseur, sur lequel la sérialisation des données pose moins de problèmes. Si le nombre de routines diffusables n'est pas en permanence au moins égal au nombre de processeurs, un ou plusieurs d'entre eux seront inactifs une partie du temps.

Le nombre de routines diffusables est :

```
Le nombre total de routines dans le système,
 moins le nombre de routines qui attendent une entrée-sortie,
 moins le nombre de routines qui attendent une ressource partagée,
 moins le nombre de routines qui attendent les résultats d'une autre routine,
 moins le nombre de routines en veille à leur demande.
```

Une charge de travail peut faire l'objet d'un *multitraitement* si elle présente en permanence autant de routines diffusables qu'il y a de processeurs dans le système. Notez qu'il ne s'agit pas du nombre *moyen* de routines diffusables. Si le nombre de routines diffusables est égal à zéro la moitié du temps, et double de celui de processeurs le reste du temps, le nombre moyen de routines diffusables est effectivement égal au nombre de processeurs, mais tous les processeurs ne sont actifs que la moitié du temps.

Augmenter le multitraitement de la charge de travail implique :

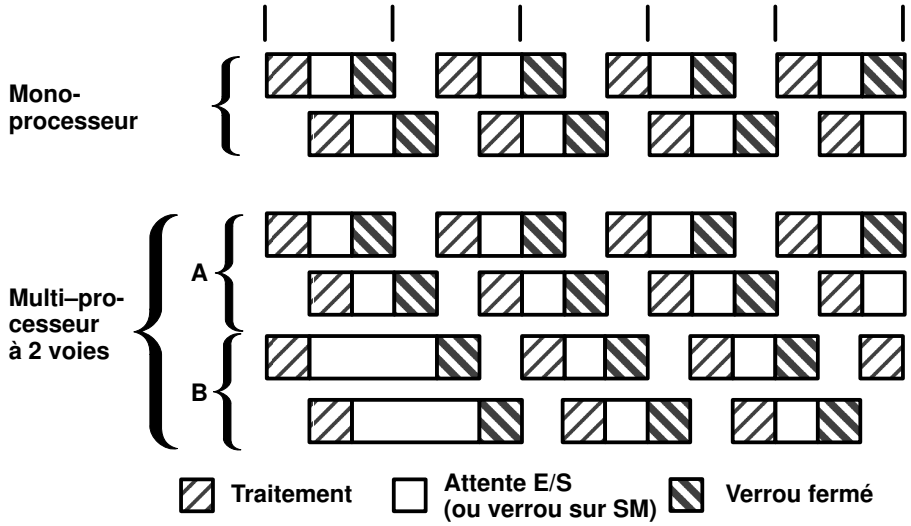
- l'identification et la suppression des goulets d'étranglement qui obligent les routines à attendre
- l'augmentation du nombre total de routines dans le système.

Ces deux solutions ne sont pas indépendantes. Si un grand goulet d'étranglement se forme sur le système, l'augmentation du nombre de routines de la charge de travail existante qui passent dans le goulet d'étranglement va accroître la proportion de routines en attente. S'il n'y a pas de goulet d'étranglement, l'augmentation du nombre de routines peut en créer un.

Échelle de débit dans un système multiprocesseur

Tous ces facteurs contribuent à ce que l'on appelle l'*échelle* d'une charge de travail. Cette échelle est le degré dont bénéficie le débit de la charge de travail du fait de la disponibilité de processeurs supplémentaires. Elle est généralement exprimée comme le quotient de la charge de travail dans un système multiprocesseur par le débit dans un système monoprocesseur comparable. Par exemple, si un système monoprocesseur traite 20 demandes par seconde et qu'un système quadriprocesseur traite 58 demandes par seconde, pour une charge de travail donnée, l'échelle est 2,9. Cette charge de travail a donc une échelle de débit élevée. Il s'agit par exemple d'une charge de travail constituée exclusivement de programmes longs intégrant de nombreux calculs complexes, comportant peu d'opérations d'entrée-sortie ou d'autre activité du noyau, et n'utilisant pas de données partagées. Dans la réalité, la plupart des charges de travail ne correspondent pas à ce modèle. L'échelle de débit est très difficile à mesurer. Lorsque c'est possible, il convient de fonder les estimations sur des mesures de charges de travail réelles.

La figure ”Échelle dans un système multiprocesseur” illustre les difficultés de cette opération. La charge de travail est constituée d’une série de commandes. Chaque commande comporte un tiers de traitement normal, un tiers d’attente d’entrée-sortie et un tiers de traitement avec un verrou fermé. Dans le système monoprocesseur, une seule commande peut être traitée à la fois, que le verrou soit ouvert ou fermé. Dans l’intervalle de temps indiqué sur la figure (égal à cinq fois le temps d’exécution de la commande), le système monoprocesseur traite 7,67 commandes.



Mise à l’échelle dans un système multiprocesseur

Dans le système multiprocesseur, deux processeurs se partagent l’exécution des programmes, mais il n’y a qu’un seul verrou. Par souci de simplicité, les conflits d’accès n’affectent que le processeur B. Au cours de la période indiquée, le système multiprocesseur traite 14 commandes. Le facteur d’échelle est ainsi de 1,83. On obtient le même résultat avec un système comprenant un nombre supérieur de processeurs, c’est pourquoi nous avons opté pour un système à deux processeurs. Le verrou est utilisé en permanence. Dans un système quadriprocesseur, l’échelle serait de 1,83 ou moins.

Dans la réalité, les programmes sont rarement aussi symétriques que les commandes de notre exemple. Rappelons également que nous n’avons pris en compte qu’un seul type de conflit : les conflits d’accès. Si nous avions tenu compte de la cohérence des mémoires cache et de l’affinité avec un processeur, l’échelle serait certainement encore inférieure.

Cet exemple démontre, s’il en était besoin, qu’il ne suffit pas d’ajouter des processeurs pour augmenter la vitesse de traitement d’une charge de travail. Il est également indispensable d’identifier et de minimiser les sources de conflits au sein des routines.

Selon certaines études, l’obtention d’échelles élevées ne pose de difficultés. Cependant, ces études se fondent sur des programmes courts et gourmands en temps processeur qui n’utilisent quasiment pas de fonctions noyau. Les résultats obtenus représentent en fait une limite supérieure de l’échelle, non la réalité.

Temps de réponse d'un système multiprocesseur

Un système multiprocesseur ne peut améliorer le temps de traitement d'un programme que dans la mesure où celui-ci peut être exécuté en plusieurs routines. Plusieurs moyens permettent d'exécuter en parallèle différentes parties d'un programme :

- Appels explicites de sous-routines **libpthreads** (ou de **fork()** dans les programmes plus anciens) pour créer plusieurs routines exécutables simultanément.
- Traitement du programme à l'aide d'un compilateur ou d'un préprocesseur de parallélisation qui détecte les séquences de code exécutables simultanément et génère les routines requises pour l'exécution en parallèle.
- Utilisation d'un module logiciel multiroutines.

Sans recours à une ou plusieurs de ces techniques, la durée d'exécution d'un programme dans un système multiprocesseur est la même que dans un système monoprocesseur comparable. En fait, elle peut même être supérieure du fait de la mobilisation du temps système par les opérations liées aux verrous et des retards liés à la répartition du programme entre plusieurs processeurs à des instants différents.

Même si les trois techniques sont exploitées, la réduction du temps de traitement est limitée par une règle dite Loi d'Amdahl :

Si une fraction *x* du temps de traitement *t* d'un programme dans un système monoprocesseur ne peut être traitée que de manière séquentielle, la réduction du temps d'exécution dans un système à *n* processeurs par rapport au temps d'exécution dans un système monoprocesseur comparable (*augmentation de la vitesse*) est donnée par l'équation :

$$\textit{Aug. vit} = \frac{\textit{temps syst. monopr}}{\textit{tps séq + tps multipr}} = \frac{t}{xt + \frac{(x-1)t}{n}} = \frac{1}{x + \frac{x}{n}}$$

$$\lim_{n \to \infty} \textit{aug. vit} = \frac{1}{x}$$

Par exemple, si un programme doit être exécuté pour 50 % séquentiellement et pour 50 % en parallèle, le facteur maximal d'amélioration du temps de réponse est inférieur à 2 (dans un système quadriprocesseur libre, ce facteur n'excède pas 1,6).

Programmation dans un système SMS

La prise en charge des routines, nouveauté d'AIX version 4.1, divise l'exécution des programmes en deux éléments :

- Un *processus* est l'ensemble des ressources physiques nécessaires à l'exécution du programme. Exemple : mémoire, accès aux fichiers, etc.
- Une *routine* est l'état d'exécution d'une instance de programme (contenu actuel du registre d'adresses d'instructions et des registres généraux). Une routine s'exécute toujours dans le cadre d'un processus et en exploite les ressources. Plusieurs routines peuvent s'exécuter au sein d'un même processus et partager ses ressources.

Dans les versions précédentes d'AIX, le programmateur de l'unité centrale diffusait les processus. Dans la version 4.1, il diffuse les routines.

Dans un environnement SMS, la prise en charge des routines facilite la mise en oeuvre à moindre coût d'applications ayant subi une adaptation totale SMS. Le traitement parallèle de plusieurs processus pour créer différents ordres d'exécution est fastidieux et onéreux car chaque processus possède ses propres ressources mémoire et requiert énormément de traitement système pour sa mise en place. La création de plusieurs routines dans un seul processus demande moins de traitement et occupe moins de mémoire.

La prise en charge des routines existe à deux niveaux :

- **libpthreads.a** dans l'environnement du programme et
- au niveau du noyau.

Traitement programmé de charges de travail migrées

La nouvelle distinction entre processus et routine est transparente pour les programmes existants. En fait, les charges de travail qui ont subi une migration directement à partir de versions précédentes d'AIX créent des processus comme auparavant. Chaque nouveau processus est créé avec une seule routine (*routine initiale*) qui est en concurrence avec les routines des autres processus pour l'accès aux ressources CPU. Les attributs par défaut de la routine initiale, associés aux nouveaux algorithmes de programmation, réduisent au minimum les modifications de la dynamique du système pour les charges de travail inchangées.

Les priorités sont manipulées à l'aide des commandes **nice** et **renice** et des commandes d'appel système **setpri** et **setpriority** , comme auparavant. Le programmateur permet l'exécution d'une routine donnée pendant une tranche de temps (10 ms généralement) au maximum avant de forcer le passage à la prochaine routine diffusable de priorité égale ou supérieure.

Variables de l'algorithme de programmation

Plusieurs variables affectent la programmation des routines. Certaines sont spécifiques de la prise en charge des routines et les autres résultent de considérations sur la programmation des processus :

Priorité	Indicateur fondamental de son degré de priorité pour l'accès au processeur.
Position dans la file d'attente du programmateur	La position d'une routine dans la file d'attente des routines diffusables reflète un certain nombre de conditions de priorité.
Politique de planification	Attribut qui détermine le devenir d'une routine en cours d'exécution à l'expiration de la tranche de temps qui lui est affectée.

Portée concurrentielle	Détermine si la routine est en concurrence uniquement avec les autres routines du processus ou avec l'ensemble des routines du système. Une routine pthread créée avec une portée concurrentielle au niveau du *processus* est planifiée par la bibliothèque alors que celles créées avec une portée *globale* sont planifiées par le noyau. Le programmateur de la bibliothèque utilise un pool de routines du noyau pour planifier les routines pthreads avec une portée concurrentielle. En règle générale, les routines pthreads doivent être créées avec une portée concurrentielle globale si elles traitent des E/S. La portée concurrentielle est utile si les synchronisations intra–processus sont nombreuses. Il s'agit d'un concept **libpthreads.a**.
Affinité avec un processeur	Effet de l'affinité avec un processeur sur les performances.

L'ensemble de ces considérations peut sembler complexe, mais il existe essentiellement trois démarches de gestion d'un processus donné :

Par défaut	Le processus a une routine dont le degré de priorité varie selon la consommation de l'unité centrale et dont la politique de programmation, SCHED_OTHER, est semblable à l'algorithme d'AIX version 3.
Contrôle au niveau du processus	Le processus peut avoir une ou plusieurs routines, mais leur politique de programmation est celle par défaut (SCHED_OTHER), ce qui permet l'utilisation des techniques d'AIX version 3 pour contrôler les valeurs nice et les priorités fixes. Toutes ces techniques affectent l'ensemble des routines du processus de la même façon. Si la commande **setpri()** est utilisée, la politique de programmation SCHED_RR est adoptée pour toutes les routines du processus.
Contrôle au niveau des routines	Une ou plusieurs routines peuvent être associées au processus. La politique de planification de ces routines est définie, selon le cas, à SCHED_RR ou à SCHED_FIFO. La priorité de chaque routine, fixe, est manipulée à l'aide de sous-routines.

Planification des variables d'environnement

Au sein de la structure **libpthreads.a**, une série de réglages permettent d'influer sur les performances de l'application. Ces variables d'environnement sont les suivantes :

- **SPINLOOPTIME**=n, où *n* est le nombre de tentatives effectuées face à un verrou occupé avant de passer à un autre processeur. *n* doit être une valeur positive.
- **YIELDLOOPTIME**=n, où *n* est le nombre de tentatives de libération du processeur avant le blocage sur un verrou occupé. *n* doit être une valeur positive. Le processeur est alloué à une autre routine du noyau, sous réserve qu'il y en ait une exécutable et dotée du niveau de priorité suffisant.
- **AIXTHREAD_SCOPE**={P|S}, où *P* indique une portée concurrentielle locale et *S* une portée concurrentielle globale. Vous devez indiquer "P" ou "S". Les accolades ne sont motivées que par la syntaxe. L'utilisation de cette variable d'environnement affecte uniquement les routines créées avec l'attribut par défaut. L'attribut par défaut est employé lorsque le paramètre *attr* de **pthread_create** a la valeur NULL.

Les variables d'environnement suivantes affectent la planification des routines pthreads créées avec une portée concurrentielle locale.

- **AIXTHREAD_MNRATIO**=p:k, où *k* est le nombre de routines du noyau qui doivent être employées pour gérer *p* routines pthreads exécutables. Cette variable d'environnement détermine le facteur d'échelle de la bibliothèque. Ce rapport est utilisé pour créer et terminer les routines pthreads.
- **AIXTHREAD_SLPRATIO**=k:p, où *k* est le nombre de routines du noyau devant être réservées pour *p* routines pthreads en veille. En règle générale, le nombre de routines du noyau nécessaires pour la prise en charge des routines pthreads en veille est limité, puisque ces routines sont en principe réveillées une à une à mesure du traitement des verrous et/ou des événements. Ceci permet d'économiser les ressources du noyau.
- **AIXTHREAD_MINKTHREADS**=n, où *n* est le nombre minimum de routines du noyau devant être utilisées. Le nombre de routines du noyau demandées par le planificateur de la bibliothèque ne descendra jamais en deçà de ce seuil. Il est possible de réclamer une routine du noyau en n'importe quel point. En général, une routine du noyau est demandée en raison de la fin d'une routine pthread.

Affinité avec un processeur et liaisons

Toutes choses égales par ailleurs, il est souhaitable de diffuser une routine sur le dernier processeur sur lequel elle a été utilisée. Ce critère de diffusion est appelé *affinité avec un processeur*. Le niveau d'affinité avec un processeur peut varier.

Le degré d'affinité le plus élevé est la *liaison* entre une routine et un processeur. Dans ce cas, la routine est diffusée uniquement sur ce processeur, même si d'autres processeurs sont disponibles. La commande et sous-routine **bindprocessor** lient la ou les routine(s) du processus spécifié à un processeur particulier.

Cette technique peut être utile pour les programmes gourmands en temps CPU qui subissent peu d'interruptions. Cependant, elle peut entraver l'exécution de programmes ordinaires car elle risque de retarder la diffusion d'une routine après une E/S jusqu'à libération du processeur auquel elle est liée. Si la routine est bloquée pendant toute la durée d'une E/S, il est peu probable que son contexte de traitement demeure dans les mémoires cache du processeur auquel elle est liée. Il est préférable de la diffuser sur le prochain processeur disponible.

Chapitre 4. Planification, conception et implantation

Pour être fonctionnel, un programme doit être performant.

Chaque programme doit satisfaire un ensemble d'utilisateurs, souvent hétérogène. Si les performances d'un programme se révèlent inacceptables pour un groupe significatif de ces utilisateurs, il ne sera pas utilisé. Un programme non utilisé ne remplit pas sa fonction première.

Le même critère vaut pour les logiciels sous licence et les applications écrites par les utilisateurs. C'est pourquoi la plupart des développeurs accordent une attention particulière aux performances de leurs programmes. Malheureusement, ils ne peuvent prévoir l'environnement et les conditions dans lesquelles ils seront exploités. La responsabilité finale des performances incombe à ceux qui sélectionnent (ou écrivent), planifient et installent les logiciels.

Ce chapitre décrit les étapes d'optimisation des performances d'un programme (acheté ou développé en interne). Le terme "programmeur" désigne, dans ce chapitre, l'administrateur système ou toute personne responsable de l'efficacité finale du programme.

Pour parvenir à des performances acceptables, il faut en définir et en quantifier le niveau dès le lancement du projet, et ne pas perdre de vue ces objectifs ni les moyens nécessaires à leur réalisation. Ces conseils semblent évidents, mais certains projets n'en tiennent délibérément pas compte. Ils pratiquent la formule "concevoir, coder, mettre au point, éventuellement documenter et, si le temps le permet, optimiser les performances".

Le seul moyen de garantir *à l'avance* des programmes efficaces et non pas seulement opérationnels, est d'intégrer dans la planification et le processus de développement des critères de performances. En effet, il est souvent plus difficile de faire une planification fiable sur des logiciels préexistants, l'installateur ne bénéficiant pas de la même marge de manœuvre que le développeur.

Le processus proposé peut sembler très lourd pour un petit programme, mais il ne faut pas oublier que l'objectif est double : le programme doit non seulement être performant, mais son intégration dans le système ne doit pas nuire aux performances des programmes installés.

Cette section traite des points suivants :

- "Identification des composants de la charge de travail"
- "Définition des objectifs de performance"
- "Évaluation des ressources requises par la charge de travail"
- "Évaluation des ressources requises par le programme"
- "Évaluation de la charge de travail à partir de celle du programme"

Autres points :

- "Conception et implantation de programmes performants"
- "Conseils pour une installation performante"

Identification des composants de la charge de travail

Que le programme soit créé en interne ou acheté tel quel, qu'il soit petit ou volumineux, le développeur, l'installateur et les utilisateurs potentiels doivent être capables de répondre à un certain nombre de questions :

- Qui est appelé à utiliser le programme ?
- Quelles seront les conditions d'exploitation du programme ?
- Selon quelle fréquence et à quel moment (heure, jour, mois, année) le programme sera-t-il utilisé ?
- Le programme devra-t-il cohabiter avec d'autres programmes ?
- Sur quel système sera-t-il exploité ?
- Quel est le volume et l'origine des données à traiter ?
- Les données créées par ou pour le programme seront-elles exploitées par d'autres moyens ?

Faute d'être élucidées lors de la conception du programme, ces questions risquent de rester sans réponse précise – les évaluations des programmeurs et des utilisateurs potentiels ayant de fortes chances d'être divergentes. Même dans le cas de figure apparemment fort simple où le programmeur et l'utilisateur sont une seule et même personne, il est nécessaire d'établir ces objectifs au préalable pour pouvoir confronter de façon rigoureuse résultats et objectifs. En outre, il est impossible d'estimer les performances requises sans une analyse détaillée du travail à effectuer.

Définition des objectifs

La définition et la quantification des objectifs de performances passent par la compréhension des besoins : utilisateurs et programmeurs ne fonderont sans doute pas leurs exigences sur les mêmes critères. Dans tous les cas, il faut définir, au minimum :

- Le temps de réponse maximum acceptable dans la plupart des cas pour chaque interaction utilisateur-machine, avec une définition de ce qu'est "la plupart des cas". Rappelons que le temps de réponse est calculé à partir du moment où l'utilisateur donne son "feu vert" pour lancer l'action jusqu'au moment où il reçoit de la machine suffisamment d'informations pour poursuivre sa tâche. Il s'agit d'un délai d'attente subjectif, fonction de l'utilisateur. Ce n'est pas le délai écoulé "entre l'entrée dans la sous-routine et la première instruction d'écriture".

  Si un utilisateur prétend n'accorder aucune importance au temps de réponse et ne s'intéresser qu'au résultat, demandez-lui si un temps d'exécution en autonome dix fois supérieur aux temps actuels est acceptable. S'il répond oui, passez aux problèmes de débit. Sinon, poursuivez avec l'utilisateur la discussion sur les temps de réponse.

- Le temps de réponse à la limite de l'acceptable le reste du temps. Autrement dit, le temps de réponse au-delà duquel l'utilisateur commence à penser que le système est hors service (ou du moins se plaint de la lenteur de la machine au point d'être réticent à l'utiliser). Il vous faut également préciser ce qu'est le "reste du temps", la minute de pointe de la journée, 1 % des interactions, etc. Ces termes sont également soumis à la subjectivité de l'utilisateur : par exemple, la dégradation du temps de réponse peut être plus coûteuse et préjudiciable à un certain moment de la journée.
- Le débit généralement requis et les heures où il se produit. Encore une fois, ce critère ne peut pas être ignoré. Par exemple : Supposons qu'un programme doit être exécuté deux fois par jour, à 10 h 00 et à 15 h 15. S'il est limité en CPU, requiert 15 minutes et est appelé à fonctionner sur un système multiutilisateur, son exécution doit faire l'objet d'une négociation préalable.
- La durée et les heures des périodes de débits maximaux.
- La variété des requêtes formulées et son évolution dans le temps.

- Le nombre d'utilisateurs par machine et le nombre total d'utilisateurs dans le cas d'une application multiutilisateur. Cette description doit indiquer les heures de connexion de ces utilisateurs, ainsi qu'une évaluation du temps de saisie au clavier, d'attente de l'exécution des requêtes et des temps de réflexion. Vous pouvez tenter de savoir si les temps de réflexion varient systématiquement en fonction de la requête suivante ou précédente.
- Les hypothèses de l'utilisateur sur les machines prévues. S'il pense à une machine en particulier, il vous faut le savoir *dès à présent.* De même, s'il a des exigences en terme de type, de taille, de coût, d'emplacement, d'interconnexion ou d'autres variables susceptibles de limiter votre marge de manœuvre, vous devez intégrer ces exigences à vos objectifs. L'estimation des résultats ne sera sans doute pas réalisée sur le système où le programme a été développé, testé ou installé initialement.

Évaluation des ressources requises par la charge de travail

A moins que vous n'ayez acheté un logiciel avec une documentation complète sur les ressources requises, cette tâche est souvent la plus délicate dans le processus de planification des performances. Il y a plusieurs raisons à cela :

- AIX propose toujours plusieurs moyens d'effectuer une tâche : Vous pouvez écrire un programme C (ou autre HLL), un script shell, un script **awk** ou **sed** , un dialogue AIXwindows, etc. Certaines techniques, qui apparaissent particulièrement adaptées au niveau de l'algorithme et de la productivité du programmeur, peuvent se révéler extrêmement coûteuses en termes de performances.

  Une des règles d'or est que plus le niveau d'abstraction est élevé, plus il faut, pour éviter toute surprise désagréable, être attentif au niveau des performances. Il convient notamment d'évaluer soigneusement les volumes de données et les itérations induites par des constructions apparemment anodines.

- Il est difficile de définir dans AIX le coût exact d'un processus pris individuellement. Le problème n'est pas seulement d'ordre technique, il est aussi d'ordre philosophique : si plusieurs instances d'un programme exécutées par plusieurs utilisateurs partagent les pages d'un texte de programme, à quel processus doivent être imputées ces pages de mémoire ? Le système d'exploitation conserve en mémoire les pages de fichiers récemment utilisées (simulant une mise en mémoire cache), les laissant à disposition des programmes qui les requièrent. L'espace monopolisé pour conserver ces données doit-il être imputé aux programmes qui vont de nouveau accéder à ces données ? La granularité de certaines mesures, telles que l'horloge système, peut entraîner des variations du temps CPU attribué aux instances successives d'un programme.

  Il existe deux approches pour traiter le problème des variations et des ambiguïtés relatives aux ressources. La première consiste à ignorer l'ambiguïté et à éliminer progressivement les sources de variation jusqu'à ce que les mesures atteignent un degré de cohérence acceptable. La seconde consiste à tenter de réaliser les mesures les plus réalistes possibles et à décrire statistiquement le résultat. Cette dernière approche est préférable car elle induit des résultats en corrélation avec diverses situations d'exploitation.

- Les systèmes AIX sont rarement dédiés à l'exécution d'une seule instance d'un programme unique. Le plus souvent, démons, activités de communication, charges de travail de multiples utilisateurs cohabitent. Ces activités sont rarement cumulatives. Par exemple, augmenter le nombre d'instances d'un programme n'entraîne parfois le chargement que de quelques pages de texte de programme, l'essentiel du programme étant déjà en mémoire. Cependant, le processus supplémentaire peut générer une concurrence plus importante au niveau des mémoires cache du processeur, si bien que non seulement les autres processus sont contraints de partager le temps processeur avec le nouveau venu, mais ils doivent *tous* subir davantage de cycles par instruction (le processeur étant ralenti par un nombre supérieur d'absences en mémoire cache).

  Il est recommandé d'effectuer une évaluation la plus réaliste possible :

- Si le programme existe, effectuez vos mesures sur l'installation en place la plus proche de vos objectifs.
- Si aucune installation ne coïncide, mettez en place une installation-test et évaluez la charge globalement.
- S'il est difficile de simuler une charge globale, mesurez chaque interaction et utilisez les résultats pour la simulation.
- Si le programme n'est pas encore créé, effectuez vos mesures sur un programme comparable qui utilise le même langage et présente une structure générale analogue. Rappelons que plus le langage est abstrait, plus la comparaison doit être effectuée minutieusement.
- Si aucun programme existant n'est suffisamment proche, élaborez un prototype des principaux algorithmes dans le langage prévu, mesurez-le et modélisez la charge de travail.
- Si, et seulement si, toute mesure se révèle impossible, il vous faut travailler par tâtonnements. Si les besoins en ressources doivent être évalués lors de la planification, le programme réel doit plus que jamais être mesuré dès que possible pendant la phase de développement.

L'estimation des ressources passe par la définition de quatre grandeurs principales (l'ordre important peu) :

Temps CPU	Coût processeur de la charge de travail
Accès disque	Taux de lectures/écritures générées par la charge
Mémoire réelle	Quantité de RAM requise par la charge de travail
Trafic réseau (LAN)	Nombre de paquets générés par la charge et nombre d'octets de données échangés

Les sections suivantes indiquent comment déterminer ces valeurs dans les divers cas de figure exposés précédemment.

Mesure des ressources requises

Si le programme réel, un programme analogue ou un prototype est disponible, le choix de la méthode dépend des facteurs suivants :

- Le système traite-t-il d'autres travaux que la charge de travail à mesurer ?
- Est-il possible d'utiliser des outils susceptibles de nuire aux performances (le système est-il en service ou actif uniquement pour effectuer les mesures) ?
- Jusqu'à quel point la charge réelle peut-elle être simulée ou observée ?

Mesure de la charge totale sur un système dédié

Ce cas de figure est idéal car il permet d'inclure le temps système et le coût de chaque processus.

Pour mesurer l'activité disque et CPU, lancez la commande **iostat.** La commande

```
$ iostat 5 >iostat.output
```

indique l'état du système toutes les 5 secondes au cours des opérations de mesure. Le premier résultat fourni par **iostat** communique les données cumulées depuis le dernier amorçage jusqu'au lancement de la commande **iostat**. Les résultats suivants concernent l'intervalle précédent (dans ce cas, les 5 dernières secondes). Voici un exemple de sortie de **iostat** exécutée sur un grand système :

```
tty:    tin        tout     cpu:     % user    % sys     % idle   % iowait
        1.2         1.6                60.2     10.8       23.4      5.6
Disks:      % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1         0.0       0.0       0.0          0         0
hdisk2         0.0       0.0       0.0          0         0
hdisk3         0.0       0.0       0.0          0         0
hdisk4         0.0       0.0       0.0          0         0
hdisk11        0.0       0.0       0.0          0         0
hdisk5         0.0       0.0       0.0          0         0
hdisk6         0.0       0.0       0.0          0         0
hdisk7         3.0      11.2       0.8          8        48
hdisk8         1.8       4.8       1.2          0        24
hdisk9         0.0       0.0       0.0          0         0
hdisk0         2.0       4.8       1.2         24         0
hdisk10        0.0       0.0       0.0          0         0
```

Pour mesurer la mémoire, utilisez **svmon**. La commande **svmon –G** fait le point sur l'utilisation globale de la mémoire. Le résultat statistique est exprimé en pages de 4 ko :

```
$ svmon -G
       m e m o r y          i n  u s e          p i n        p g  s p a c e
 size  inuse  free  pin   work  pers  clnt   work  pers  clnt   size  inuse
24576  24366   210 2209  15659  6863  1844   2209     0     0   40960  26270
```

Dans cet exemple, les 96 Mo de mémoire de la machine sont entièrement utilisés. 64 % de mémoire RAM environ sont dédiés aux segments de travail (lecture/écriture des programmes exécutés). S'il existe des processus longs qui vous intéressent, examinez en détail leurs besoins en mémoire. L'exemple suivant détermine la mémoire utilisée par l'un des processus de l'utilisateur xxxxxx.

```
$ ps -fu xxxxxx
    USER   PID  PPID   C    STIME    TTY  TIME CMD
  xxxxxx 28031 51445  15 14:01:56  pts/9   0:00   ps -fu xxxxxx
  xxxxxx 51445 54772   1 07:57:47  pts/9   0:00   -ksh
  xxxxxx 54772  6864   0 07:57:47      -   0:02   rlogind

$ svmon -P 51445
  Pid                          Command        Inuse        Pin      Pgspace
51445                             ksh         1668          2         4077

Pid:  51445
Command:  ksh

Segid  Type  Description        Inuse  Pin  Pgspace   Address Range
 8270  pers  /dev/fslv00:86079      1    0        0    0..0
 4809  work  shared library      1558    0     4039    0..4673 :60123..65535
 9213  work  private              37    2       38    0..31 : 65406..65535
  8a1  pers  code,/dev/hd2:14400   72    0        0    0..91
```

Le coût de cette instance de **ksh** est imputable au segment de travail (9213) dont 37 pages sont en cours d'utilisation. Le coût des 1558 pages de la bibliothèque partagée et celui des 72 pages de l'exécutable **ksh** est réparti respectivement sur l'ensemble des programmes et des instances de **ksh**.

Si vous estimez que la taille du système de 96 Mo est excessive, utilisez la commande **rmss** pour réduire la taille effective de la machine et reprenez les mesures de la charge. Si les opérations de pages augmentent sensiblement ou que le temps de réponse s'allonge, cela signifie que vous avez trop réduit la mémoire. Appliquez cette méthode jusqu'à ce que vous trouviez la taille la plus appropriée à l'exécution de la charge dans les meilleures conditions. Pour en savoir plus sur cette méthode, reportez-vous à "Estimation de la mémoire requise via rmss", page 7-6.

La principale commande pour mesurer l'utilisation du réseau est **netstat**. L'exemple ci-dessous illustre l'activité d'une interface de réseau en anneau à jeton :

```
$ netstat -I tr0 5
   input    (tr0)     output           input   (Total)    output
 packets  errs  packets  errs colls   packets  errs  packets  errs colls
35552822 213488 30283693    0     0  35608011 213488 30338882    0     0
     300      0      426     0     0       300      0      426     0     0
     272      2      190     0     0       272      2      190     0     0
     231      0      192     0     0       231      0      192     0     0
     143      0      113     0     0       143      0      113     0     0
     408      1      176     0     0       408      1      176     0     0
```

La première ligne indique le trafic cumulé sur le réseau depuis le dernier amorçage. Chaque ligne qui suit fournit l'activité enregistrée lors des 5 secondes précédentes.

Mesure de la charge totale sur un système actif

La méthode à utiliser est analogue à celle appliquée aux systèmes dédiés, mais vous devez, dans ce cas, veiller à préserver les performances du système. Il faut savoir, par exemple, que le coût d'exécution de la commande **svmon –G** est très élevé. L'annexe E, "Outils de performance", donne une estimation des coûts, en termes de ressources, des outils de performances les plus couramment utilisés.

L'outil le moins coûteux est probablement **vmstat**, qui fournit des informations sur la mémoire, les opérations d'E/S et l'utilisation du processeur, en un seul rapport. Si les intervalles entre deux rapports **vmstat** restent raisonnablement longs (10 secondes), le coût moyen d'utilisation de cette commande est faible. Pour plus de détails sur l'utilisation de **vmstat**, reportez-vous à "Identification des ressources limitatives", page 12-23.

Mesure d'une charge partielle sur un système actif

Une telle mesure est intéressante lorsque vous souhaitez transférer ou dupliquer une partie de la charge pour l'exécuter sur un autre système. Le système étant en service, votre intervention doit l'affecter le moins possible. Il vous faut analyser la charge en détail pour distinguer la partie qui vous intéresse et déterminer les éléments communs. Il peut s'agir :

- du programme ou de quelques programmes associés,
- du travail effectué par certains utilisateurs du système,
- du travail provenant de certains terminaux.

Selon le critère choisi, utilisez l'une des commandes :

```
ps -ef | grep pgmname
ps -fuusername, . . .
ps -ftttyname, . . .
```

pour identifier les processus qui vous intéressent et de faire le point sur leur consommation en temps CPU. Pour évaluer l'occupation mémoire de ces processus, vous pouvez utiliser la commande **svmon** (judicieusement !).

Mesure d'un programme particulier

Plusieurs outils sont disponibles pour mesurer la consommation en ressources de programmes particuliers. Certains offrent une mesure de la charge plus complète que d'autres, mais ils ont trop d'incidence sur le système pour être appliqués à un système en service. La plupart de ces outils sont décrits dans les chapitres relatifs à l'optimisation de la consommation de ressources spécifiques. Les principaux sont :

time	mesure le délai d'exécution écoulé et la consommation CPU d'un programme individuel. Reportez-vous à "Mesure de la CPU via time", page 6-3.
tprof	mesure la consommation relative de CPU par les programmes, les bibliothèques de sous-routines et le noyau AIX. Reportez-vous à "Analyse des programmes via tprof", page 6-10.
svmon	mesure la mémoire réelle utilisée par un processus. Reportez-vous à "Quantité de mémoire utilisée", page 7-2.
vmstat –s	permet de mesurer la charge des E/S générée par un programme. Reportez-vous à "Mesure des E/S disque globales via vmstat", page 8-7.

Estimation des ressources requises par un nouveau programme

Il est impossible d'évaluer précisément des programmes qui n'ont pas encore été écrits : les adaptations et nouveautés apportées à un programme pendant la phase de codage sont imprévisibles. Toutefois, quelques règles de base peuvent vous aider à faire une évaluation approximative. Au minimum, un programme requiert au départ :

- Temps CPU
  - environ 50 millisecondes, en général de temps système,
- Mémoire réelle
  - une page par texte de programme,
  - une quinzaine de pages (dont 2 fixes) pour le segment (de données) de travail,
  - un accès à **libc.a** (généralement partagé entre tous les programmes et englobé dans le coût de base du système d'exploitation),
- E/S disque
  - environ 12 opérations de chargement de pages si le programme n'a pas été compilé, copié ou récemment utilisé, sinon 0.

Ajoutez les coûts représentés par les demandes liées à la conception (les temps CPU indiqués concernent le modèle 660) :

- Temps CPU
  - La consommation CPU d'un programme ordinaire, contenant peu de niveaux d'itérations ou d'appels de sous-routines coûteuses, est négligeable et quasi-impossible à mesurer.
  - Si le programme fait appel à des algorithmes de calcul lourds, chaque algorithme doit être mesuré à partir d'un prototype.
  - Si le programme fait appel à des sous-routines de bibliothèques coûteuses en calculs (éléments de X ou de Motif ou **printf**, par exemple), mesurez la consommation CPU en vous basant sur d'autres programmes ordinaires.

- Mémoire réelle
  - Comptez (*très* approximativement) 350 lignes de code par page de texte programme. C'est-à-dire environ 12 octets par ligne. N'oubliez pas que le style de code et les options du compilateur peuvent multiplier ou diviser le résultat par deux.
    Cette variation concerne les pages impliquées dans le scénario type.
    Si, occasionnellement, des sous-routines exécutées sont insérées à la fin de l'exécutable, les pages concernées ne sollicitent généralement pas la mémoire réelle.
  - Les références à des bibliothèques partagées autres que **libc.a** n'augmentent la consommation mémoire que dans la mesure où ces bibliothèques ne sont pas partagées par d'autres programmes ou instances du programme considéré. Pour mesurer la taille des bibliothèques, écrivez un programme simple, d'exécution longue qui y fait référence, et appliquez **svmon –P** au processus.
  - Évaluez la capacité de stockage requise par les structures de données identifiées dans le programme. Arrondissez le résultat à la page la plus proche.
  - Dans une exécution courte, chaque opération d'E/S disque utilisera une page de mémoire. Supposons que la page doit déjà être disponible. Ne supposez pas que le programme va attendre la libération d'une autre page du programme.
- E/S disque
  - Pour les E/S séquentielles, chaque lot de 4 096 octets lus ou écrits génère une opération d'E/S, sauf si certaines pages du fichier sont encore en mémoire du fait d'une utilisation récente.
  - Pour les E/S aléatoires, chaque accès, même limité, à une page différente de 4 096 octets génère une opération d'E/S, sauf si certaines pages du fichier sont encore en mémoire du fait d'une utilisation récente.
  - Dans des conditions de laboratoire, chaque lecture ou écriture séquentielle d'une page de 4 ko dans un fichier volumineux mobilise environ 140+/–20 microsecondes de temps CPU. Chaque lecture ou écriture aléatoire d'une page de 4 ko mobilise environ 350+/–40 microsecondes de temps CPU. Rappelons qu'en pratique les fichiers ne sont pas nécessairement stockés séquentiellement sur disque, même s'ils sont écrits et lus séquentiellement dans le programme. Par conséquent, un accès disque représente en réalité un coût CPU généralement plus proche de celui d'un accès aléatoire que d'un accès séquentiel théorique.
- E/S de communication
  - Les E/S disque sur des systèmes de fichiers montés à distance, de type NFS ou AFS, sont exécutées sur le serveur. Dans ce cas, il y a davantage de demandes mémoire et CPU côté client.
  - Les appels de procédure à distance (RPC) de tous types constituent une bonne part de la charge CPU. C'est pourquoi il est conseillé de les réduire au minimum, de différer leur traitement, d'en faire un prototype et de les mesurer à l'avance.
  - Dans des conditions de laboratoire, chaque lecture ou écriture NFS d'une page de 4 ko mobilise environ 670+/–30 microsecondes de temps CPU client.
    Chaque lecture ou écriture aléatoire NFS d'une page de 4 ko mobilise environ 1000+/–200 microsecondes de temps CPU client.

Évaluation de la charge de travail à partir de celle du programme

Pour évaluer les besoins en ressources moyens et maximaux, le mieux est d'utiliser un modèle de mise en file d'attente du type BEST/1. Si vous optez pour un modèle statique, vous risquez de sous ou surestimer la limite maximale. Dans les deux cas, vous devez bien appréhender les interactions entre programmes, en termes de ressources.

Si vous élaborez un modèle statique, utilisez l'intervalle de temps correspondant au pire temps de réponse acceptable pour le programme le plus sollicité. Déterminez, en fonction du nombre d'utilisateurs prévu, leur temps de réflexion, la fréquence des saisies au clavier, les diverses opérations par anticipation et les programmes généralement exécutés pendant chaque intervalle.

- Temps CPU
  - Faites la somme des besoins CPU requis par tous les programmes exécutés durant cet intervalle. N'oubliez pas d'y inclure les temps requis par les E/S de communication et les E/S disque effectuées par les programmes.
  - Si ce nombre est supérieur à 75 % du temps CPU disponible durant cet intervalle, envisagez de réduire le nombre d'utilisateurs ou d'augmenter la capacité CPU.
- Mémoire réelle
  - Commencez avec 6 à 8 Mo pour le système d'exploitation lui-même. Le chiffre inférieur concerne un système autonome. Le dernier est applicable à un système connecté au LAN et exploitant TCP/IP et NFS.
  - Faites la somme des besoins des segments de travail pour toutes les instances de programmes qui doivent être exécutées durant cet intervalle, en incluant l'espace estimé pour les structures de données du programme.
  - Ajoutez la capacité mémoire requise du segment de texte pour chaque programme distinct devant être exécuté (une copie d'un texte de programme dessert toutes les instances de ce programme). Rappelons que toutes les sous-routines (et seulement les sous-routines) provenant de bibliothèques non partagées sont partie intégrante de l'exécutable, mais que les bibliothèques ne sont pas en mémoire.
  - Ajoutez la quantité d'espace consommé par chacune des bibliothèques partagées utilisées par les programmes. Ici aussi, une copie dessert tous les programmes.
  - Pour réserver suffisamment d'espace à la mise en mémoire cache de certains fichiers et à la liste ouverte, vos prévisions en mémoire totale ne doivent pas excéder 80 % de la mémoire de la machine prévue.
- E/S disque
  - Faites la somme des E/S impliquées par chaque instance de chaque programme. Distinguez entre les E/S sur les petits fichiers et les E/S aléatoires sur les grands fichiers, d'une part, et les E/S purement séquentielles sur les gros fichiers (de plus de 32 ko), d'autre part.
  - Retranchez les E/S qui, selon vous, seront satisfaites à partir de la mémoire. Tout enregistrement lu ou écrit dans l'intervalle précédent est probablement toujours disponible dans l'intervalle courant. Par ailleurs, comparez la taille de la machine considérée avec la capacité RAM totale requise pour la charge de travail. L'espace restant, une fois satisfaits les besoins du système d'exploitation et de la charge de travail, contient probablement les pages de fichiers récemment lues ou écrites. Si votre application est conçue de sorte que la probabilité de réutiliser les données récemment exploitées est très élevée, vous pouvez revoir à la baisse les prévisions d'utilisation de la mémoire cache. Notez que la réutilisation des données se fait au niveau de la page et non de l'enregistrement. S'il est peu probable qu'un enregistrement soit réutilisé, mais qu'une page contienne de nombreux enregistrements, certains des enregistrements requis pendant un intervalle donné ont des chances de se trouver sur la même page que d'autres enregistrements récemment utilisés.

  - Comparez le niveau net d'E/S requis à la capacité approximative des unités de disques courantes (voir le tableau). Si la capacité requise en pages aléatoires et séquentielles est supérieure à 75 % de la capacité totale des disques porteurs des données de l'application, il vous faudra certainement optimiser, voire étendre ces capacités lors de l'exécution de l'application.

- E/S de communication
  - Calculez l'utilisation de la largeur de bande de la charge. L'utilisation totale de la largeur de bande de tous les nœuds du réseau LAN ne doit pas excéder 70 % de la largeur de bande nominale (50 % pour Ethernet).
  - Il est conseillé d'effectuer la même analyse sur les besoins CPU, mémoire et E/S de la charge supplémentaire que le serveur devra prendre en charge.

Rappelons que ces mesures ne fournissent qu'une estimation très approximative et qu'il est préconisé de remplacer dès que possible les résultats estimés par une mesure réelle : l'estimation globale n'en sera que plus précise.

Conception et implantation de programmes performants

Si vous avez déjà localisé la ressource qui ralentit votre programme, passez directement à la section traitant des moyens de minimiser son utilisation. Sinon, vous devez envisager d'équilibrer votre programme, en vous aidant des recommandations de ce chapitre. Une fois le programme installé, passez à "Identification des ressources limitatives", page 1-1.

Cette section traite des points suivants :

- "Programmes à CPU limitée"
- "Conception et codage pour l'exploitation optimale des mémoires cache"
- "Préprocesseurs et des compilateurs XL"
- "Programmes à CPU limitée"

Programmes à CPU limitée

La vitesse maximale d'un programme réellement limité en temps processeur est fonction :

- de l'algorithme utilisé,
- du code source et des structures de données créées par le programmeur,
- de la séquence d'instructions en langage machine générée par le compilateur,
- de la taille et des structures des mémoires cache du processeur,
- de l'architecture et de la vitesse d'horloge du processeur.

Si le programme est limité en CPU simplement parce qu'il est constitué quasi-exclusivement d'instructions de calcul numérique, l'algorithme choisi sera évidemment déterminant pour les performances du programme. Le débat sur le choix des algorithmes dépasse le cadre de ce manuel. On suppose que les performances de calcul ont été prises en compte dans le choix de l'algorithme.

L'algorithme étant donné, les seuls éléments de la liste sur lesquels le programmeur peut intervenir sont le code source, les options de compilation et éventuellement les structures de données. Les sections qui suivent présentent les techniques d'amélioration de programmes à partir du code source. Si l'utilisateur ne dispose pas du code source, il doit recourir aux techniques d'optimisation ou de gestion de la charge.

Conception et codage pour l'exploitation optimale des mémoires cache

Comme indiqué à "Performances", les processeurs du ESCALA disposent d'une mémoire hiérarchique multiniveau :

1. Pipeline d'instructions et registres CPU
2. Mémoires cache d'instructions et de données avec tampons TLB correspondants
3. RAM
4. Disque

Au fur et à mesure de leur progression dans la hiérarchie, les instructions et les données transitent par des mémoires de plus en plus rapides, mais aussi de plus en plus réduites et consommatrices. Pour tirer le meilleur parti d'une machine, le programmeur doit s'attacher à exploiter au mieux la mémoire disponible à chaque niveau de la hiérarchie.

Pour cela, la règle de base consiste à maintenir en mémoire les instructions et données susceptibles d'être utilisées. Mais une difficulté doit être surmontée : la mémoire est affectée en blocs de longueur fixe (en lignes de cache ou pages de mémoire réelle, par exemple) qui ne correspondent généralement pas aux structures des programmes ou des données. Cette inadéquation diminue l'efficacité des mémoires affectées, et par voie de conséquence, les performances des systèmes, notamment de petite taille ou particulièrement chargés.

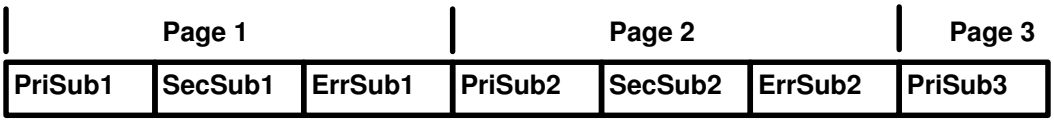
Prendre en compte la hiérarchie des mémoires ne signifie pas programmer en fonction d'une page ou d'une taille de ligne de cache. Il s'agit simplement de comprendre et d'adapter les règles générales de programmation à un environnement de mémoire virtuelle ou cache. Des techniques de redécoupage des modules permettent d'obtenir des résultats satisfaisants sans modifier le code existant. Naturellement, si du code doit être ajouté, il faut tenir compte des impératifs d'efficacité de mémoire.

En terme d'optimisation des mémoires hiérarchiques, deux notions sont incontournables : le "regroupement référentiel" et la "partie active". Le *regroupement référentiel* d'un programme est le degré de regroupement de ses adresses d'exécution d'instruction et des références de données dans une zone restreinte de la mémoire pendant un intervalle donné.
La *partie active* d'un programme, au cours du même intervalle, est l'ensemble des blocs mémoire utilisés, ou, plus précisément, le code ou les données qui occupent ces blocs. Un programme présentant un bon regroupement référentiel aura une partie active minimale, les blocs utilisés étant étroitement associés aux codes et aux données. Inversement, la partie active d'un programme équivalent au niveau fonctionnel mais présentant un regroupement référentiel médiocre sera plus étendue, davantage de blocs étant nécessaires pour accéder à un éventail d'adresses plus étendu.

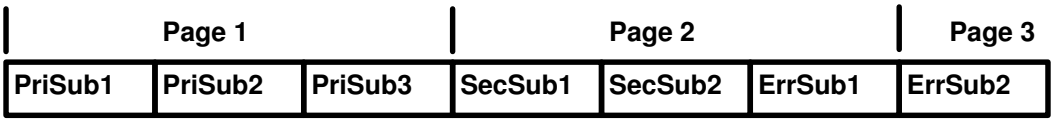
Le chargement de chaque bloc à un niveau donné de la hiérarchie demandant un temps non négligeable, une programmation efficace sur un système à mémoire hiérarchique doit viser à limiter au minimum la partie active du programme par un codage approprié.

La figure "Regroupement référentiel", présente deux schémas. Le premier, déconseillé, illustre un programme vraisemblablement structuré dans l'ordre dans lequel il a été écrit, en quelque sorte en suivant le "fil de la pensée" : La première sous-routine, **PriSub1**, contient le point d'entrée du programme. Il utilise toujours les sous-routines principales **PriSub2** et **PriSub3**. Certaines fonctions peu utilisées du programme demandent des sous-routines secondaires **SecSub1** et **2**. Très rarement, les sous-routines d'erreur **ErrSub1** et **2** sont requises. Cette structuration offre un regroupement référentiel médiocre, sollicitant trois pages de mémoire en exécution normale. Les sous-routines secondaires et les sous-routines d'erreur divisent le chemin principal du programme en trois sections physiquement distantes.

Faible regroupement référentiel et partie active étendue



Fort regroupement référentiel et partie active réduite



Regroupement référentiel

Le second schéma propose une version améliorée du programme où les sous-routines principales sont regroupées, les fonctions peu sollicitées placées ensuite et les sous-routines d'erreur, nécessaires mais peu utilisées, placées en fin de programme. Les fonctions les plus courantes peuvent à présent être traitées en une seule lecture disque et une seule page de mémoire au lieu des trois du schéma précédent.

Rappelons que le regroupement référentiel et la partie active sont définis par rapport au temps. Si un programme fonctionne par étape prenant chacune un certain temps et utilisant différentes sous-routines, il est conseillé de limiter au minimum la partie active à chaque étape.

Registres et pipeline

En général, l'affectation et l'optimisation de l'espace de registre, ainsi que le maintien des données dans le pipeline sont à la charge des compilateurs. En revanche, il incombe au programmeur d'éviter toute structure en contradiction avec les mécanismes d'optimisation du compilateur. Par exemple, si vous insérez une sous-routine dans l'une des boucles critiques, le compilateur préférera sans doute l'incorporer dans la partie principale du programme pour limiter le temps d'exécution. Or, si cette sous-routine a été placée dans un module **.c** différent, le compilateur ne pourra la déplacer.

Cache et tampon TLB

En fonction de l'architecture (POWER, POWER 2 ou PowerPC) et du modèle, les processeurs du ESCALA ont un ou plusieurs caches à prendre en charge :

- Parties de programmes d'exécution,
- Données utilisées par ces programmes,
- Tampons TLB (Translation Lookaside Buffer) contenant les informations de mappage entre les adresses virtuelles et les adresses réelles des pages de texte d'instructions ou de données récemment utilisées.

Si une absence en cache se produit, le chargement d'une ligne complète de cache peut demander plus d'une douzaine de cycles processeur. En cas d'absence en TLB, le calcul du mappage virtuel-réel d'une page peut demander plusieurs douzaines de cycles. Le coût exact de ces opérations dépend de l'implantation. Pour en savoir plus sur les architectures des mémoires cache, reportez-vous à l'annexe C.

Même si un programme et ses données tiennent dans les mémoires cache, plus il y a de lignes ou d'entrées TLB utilisées (c'est–à–dire, plus médiocre est le regroupement référentiel), plus le nombre de cycles CPU augmente pour mener à bien le chargement. Sauf si les instructions et les données sont fréquemment réutilisées, la surcharge liée au chargement constitue une part importante du temps d'exécution du programme, entraînant des performances moindres du système.

Sur les machines équipées de mémoires cache, il est conseillé de maintenir la partie principale du programme contenant les opérations courantes la plus compacte possible. De plus, la procédure principale et l'ensemble des sous-routines qu'elle appelle doivent être contigus. Les opérations plus exceptionnelles, telles que les erreurs inexpliquées, doivent seulement être testées dans la partie principale. Si cette condition intervient réellement, elle doit être traitée dans une sous-routine distincte. Toutes les sous-routines de ce type doivent être regroupées à la fin du module. Cette structure permet d'éviter que des codes peu utilisés monopolisent de l'espace dans les lignes cache très sollicitées. On peut même imaginer dans le cas de modules importants que certaines, voire la totalité, des sous-routines peu usitées occupent une page qui n'est pratiquement jamais lue en mémoire.

Le même principe est applicable aux structures de données, même s'il est parfois nécessaire de modifier le code source du fait des règles d'organisation de données par les compilateurs. Ce type de difficulté a été rencontré durant le développement d'AIX version 3. Certaines opérations matricielles, telle la multiplication, impliquent des algorithmes qui, s'ils ont été codés de façon trop simpliste, présentent un regroupement référentiel médiocre. Ces opérations supposent généralement un accès séquentiel aux données de la matrice (éléments de rangées agissant sur des éléments de colonnes, par exemple). Chaque compilateur a ses propres règles de stockage des matrices. Par exemple, le compilateur XL FORTRAN les organise par colonnes (éléments de la colonne 1 suivis des éléments de la colonne 2, etc.), mais le compilateur XL C les organise par rangées. Si les matrices sont de petite taille, les éléments des rangées et des colonnes peuvent être placés dans la mémoire cache de données, et le processeur et l'unité à virgule flottante s'exécuter à la vitesse maximale. Mais, plus la taille des matrices augmente, plus le regroupement référentiel de ces opérations de rangées/colonnes se détériore, jusqu'au point où les données ne peuvent plus être stockées dans la mémoire cache.

En fait, le modèle ”naturel” d’accès des opérations sur les rangées/colonnes génère un mécanisme d’emballement de la mémoire : il y a accès à une chaîne d’éléments de taille supérieure au cache, si bien que les éléments utilisés initialement sont expulsés et la même méthode d’accès de nouveau appliquée pour les mêmes données. La solution généralement préconisée est de partitionner l’opération en blocs, de sorte que les diverses opérations portant sur les mêmes éléments puissent être exécutées quand les éléments sont encore en mémoire cache. Cette technique générale est appelée *exploitation par blocs*. Un groupe de chercheurs compétents en analyse numérique a été sollicité pour coder des versions des algorithmes de manipulation de matrices, en utilisant l’exploitation par blocs et d’autres techniques d’optimisation. Résultat : des performances 30 fois meilleures pour la multiplication des matrices. Cette version optimisée des routines a été installée dans la bibliothèque BLAS (Basic Linear Algebra Subroutines) d’AIX, **/usr/lib/libblas.a**. Un jeu plus étendu de sous-routines optimisées est également disponible dans le programme sous licence ESSL (Engineering and Scientific Subroutine Library), documenté dans le manuel *IBM Engineering and Scientific Subroutine Library Guide and Reference*.

Les fonctions et les interfaces des sous-routines BLAS (Basic Linear Algebra Subroutines) sont décrites dans *AIX Technical Reference, Volume 2: Base Operating System and Extensions*. Cette bibliothèque n’est accessible que si l’environnement d’exécution FORTRAN est installé. Elle est destinée particulièrement aux utilisateurs qui doivent effectuer leurs propres opérations vectorielles et matricielles :les sous-routines proposées offrent un degré d’optimisation tel qu’un non spécialiste d’analyse numérique ne risque pas de faire mieux.

Si le programmeur contrôle les structures des données, d’autres améliorations peuvent être envisagées. En général, il est préconisé de regrouper au maximum les données souvent utilisées. Si une structure contient à la fois des informations de contrôle très sollicitées et des données détaillées peu sollicitées, assurez-vous que les informations de contrôle sont affectées sur des octets consécutifs. Cette précaution permettra de charger la totalité des informations de contrôle en mémoire cache en limitant les cas d’absences en cache (à quelques-uns, voire à un seul).

Préprocesseurs et compilateurs XL

Pour exploiter au mieux un programme sur une machine donnée, il faut savoir que :

- Certains préprocesseurs permettent de réorganiser la structure de certains codes source de façon à obtenir un module source, équivalent au niveau fonctionnel, mais compilable en code exécutable plus performant.
- Plusieurs options de compilation sont disponibles en fonction de la ou des variantes d’architecture POWER utilisées.
- La fonction **#pragma** permet de signaler certains aspects du programme au compilateur XL C pour qu’il génère un code plus efficace en allégeant certaines hypothèses concernant les cas les moins favorables.
- Plusieurs niveaux d’optimisation confèrent au compilateur une marge de liberté plus ou moins étendue pour réorganiser les instructions.

Pour les programmeurs pressés, une seule règle est préconisée : *l’optimisation systématique*. Le gain de performances induit par une optimisation du code, aussi simple soit-elle, est tel qu’il serait aberrant d’en faire l’économie (ne serait-ce qu’en appliquant l’option **–O** de la commande **cc**, **xlc** ou **xlf**). Les seules exceptions à cette règle sont les conditions de test qui exigent de générer le code tel quel (par exemple, pour analyser les performances au niveau des instructions par la fonction **tprof**).

Les autres techniques d’optimisation confèrent à certains programmes des performances supérieures mais il faut procéder à nombre de recompilations et mesures pour déterminer la combinaison technique la mieux appropriée à un programme particulier.

Les différentes techniques d'exploitation optimales des compilateurs sont présentées dans les sections qui suivent. Pour une description plus détaillée, reportez-vous au manuel *Optimization and Tuning Guide for XL Fortran, XL C and XL C++*.

Préprocesseurs de code source

Il existe plusieurs préprocesseurs de code source disponibles pour le ESCALA. Trois nous intéressent pour le moment :

- KAP/C (de Kuck and Associates)
- KAP/FORTRAN (de Kuck and Associates)
- VAST (de PSR)

Une des techniques utilisées par ces préprocesseurs est la reconnaissance des codes, technique mathématiquement équivalente à celle appliquée par les sous-routines des bibliothèques ESSL ou BLAS citées précédemment : le préprocesseur remplace le code de calcul initial par un appel à une sous-routine optimisée. Les préprocesseurs tentent également de modifier les structures de données pour les adapter au mieux aux machines ESCALA.

Compilation propre à une architecture

L'option de compilation **–qarch** permet d'indiquer l'architecture POWER (POWER, POWER 2 ou PowerPC) sur laquelle l'exécutable sera exploité. Les valeurs possibles sont :

–qarch=COM	Compilation du sous-ensemble commun des trois jeux d'instructions. Les programmes compilés avec cette option fonctionnent correctement sur les trois architectures. Valeur par défaut.
–qarch=PWR	Compilation pour l'architecture POWER du ESCALA d'origine. Les programmes compilés avec cette option fonctionnent correctement sur les trois architectures, mais certaines instructions seront simulées sur les systèmes PowerPC, au détriment des performances.
–qarch=PWRX	Compilation pour POWER2. Les programmes qui utilisent largement la racine carrée à virgule flottante simple ou double précision offriront des performances supérieures. Les exécutables ne doivent être exploités que sur des systèmes POWER2.
–qarch=PPC	Compilation pour PowerPC. Les programmes qui font souvent appel à l'extraction de racines carrées à virgule flottante simple offriront des performances supérieures. L'exécutable ne doit être exploité que sur des systèmes PowerPC.

L'option de compilation **–qtune** oriente le compilateur sur l'architecture à favoriser. A la différence de **–qarch**, **–qtune** ne génère pas d'instructions propres à une architecture. L'option indique seulement au compilateur, dans le cas d'une alternative, la technique la plus appropriée à une architecture. Les valeurs possibles de **–qtune** sont :

–qtune=PWR	Le programme fonctionne essentiellement sur des systèmes POWER.
–qtune=PWRX	Le programme fonctionne essentiellement sur des systèmes POWER2.
–qtune=601	Le programme fonctionne essentiellement sur des systèmes PowerPC 601.

La figure "Combinaisons des valeurs de –qarch et –qtune" indique les combinaisons de valeurs possibles et par défaut de l'option **–qtune** pour les valeurs spécifiées de **–qarch**. Si aucune option n'est spécifiée, **–qarch=COM –qtune=PWR** est appliqué par défaut.

	–qtune=PWR	–qtune=PWRX	–qtune=601	par défaut
–qarch=COM	**OK**	**OK**	**OK**	**PWR**
–qarch=PWR	**OK**	**OK**	**OK**	**PWR**
–qarch=PWRX	**OK**	**OK**	**non valide**	**PWRX**
–qarch=PPC	**OK**	**non valide**	**OK**	**601**

Combinaisons des valeurs de –qarch et –qtune

Directive #pragma

Dans certains cas, l'optimiseur peut être inhibé par la nécessité de générer un code qui tienne compte des conditions les moins favorables. Pour signaler au compilateur que certaines contraintes peuvent être assouplies et autoriser ainsi la génération d'un code plus efficace, vous disposez de la directive **#pragma**.

Un *pragma* est une instruction définie à l'implantation. Les pragmas sont de la forme :

```
#pragma character_sequence ...
```

Les pragmas en XL C ci-dessous peuvent sensiblement améliorer les performances d'un programme sans modifier le reste du code source :

- `disjoint`
- `isolated_call`

Directive #pragma disjoint

La directive **#pragma disjoint** répertorie les identificateurs qui, dans le cadre de leur utilisation, ne jouent pas le rôle d'alias entre eux :

```
#pragma disjoint ( { identifier | *identifier }
                 [,{ identifier | *identifier } ] ... )
```

La directive informe le compilateur qu'aucun des identificateurs de la liste ne partage la même mémoire physique, ce qui donne plus de latitude pour les optimisations. Si l'un des identificateurs de la liste partage en réalité de la mémoire physique, les résultats du programme risquent d'être incorrects.

Cette directive peut apparaître à n'importe quel point du programme source.
Un identificateur doit être cité dans le programme là où la directive apparaît. Les identificateurs ne peuvent renvoyer à :

- un membre d'une structure ou d'une union,
- un lien à une structure ou une union,
- une constante d'énumération,
- une étiquette.

Les identificateurs doivent être déclarés avant d'être cités dans une pragma. Aucun pointeur de la liste des identificateurs ne doit avoir servi d'argument à une fonction avant d'apparaître dans la directive.

L'utilisation de la directive **#pragma disjoint** est illustrée dans l'exemple ci-dessous. Le pointeur externe `ptr_a` ne partage pas de mémoire avec la variable externe `b` pas plus qu'il ne pointe vers elle, le compilateur peut donc en déduire que la proposition 7 pour l'objet sur lequel `ptr_a` pointe ne modifiera pas la valeur `b`. De même, le pointeur externe `ptr_b` ne partage pas de mémoire avec la variable externe `a`, pas plus qu'il ne pointe vers elle. Par conséquent, le compilateur peut en déduire que l'argument de `another_function` prend la valeur 6.

```
int a, b, *ptr_a, *ptr_b;

#pragma disjoint(*ptr_a, b)    /* ptr_a ne pointe jamais sur b */
#pragma disjoint(*ptr_b, a)      /* ptr_b ne pointe jamais sur a
*/
one_function()
{
    b = 6;
    *ptr_a = 7; /* La proposition ne changera pas la valeur b */
    another_function(b);     /* L'argument "b" a pour valeur 6 */
}
```

#pragma isolated_call

La directive **#pragma isolated_call** répertorie les fonctions qui ne modifient pas les objets de données visibles lors de l'appel de la fonction.

```
#pragma isolated_call ( identifier [ , identifier ] ... )
```

Le pragma doit être placé avant tout appel aux fonctions de la liste d'identificateurs. Les identificateurs de la liste doivent être déclarés avant d'être cités dans une pragma. Toute fonction répertoriée dans la liste et appelée avant que la directive ne soit utilisée n'est pas traitée comme un appel isolé. Les identificateurs doivent être de type fonction ou de type définition de fonction.

La directive indique au compilateur qu'aucune fonction citée n'induit d'effets secondaires. Par exemple, l'accès à un objet volatile, la modification d'un objet externe, la modification d'un fichier ou l'appel d'une fonction exécutant l'une de ces opérations peut être considéré comme exempt d'effet secondaire. Toute modification d'état de l'environnement d'exécution peut être considérée comme un effet secondaire. Le passage d'arguments de fonction par référence est l'un des effets secondaires autorisés, mais il faut savoir que les effets secondaires peuvent générer des résultats erronés lorsqu'ils apparaissent dans les directives **#pragma isolated_call**.

Déclarer une fonction comme isolée indique à l'optimiseur que les variables externes et statiques ne peuvent pas être modifiées par la fonction appelée et que, le cas échéant, les références mémoire peuvent être supprimées à partir de la fonction appelée. Il est possible d'agencer les instructions plus librement de façon à limiter les délais de pipelines et à accélérer l'exécution. Il faut savoir que réagencer les instructions peut générer un code qui requiert davantage de valeurs de portée générale et/ou de registres à virgule flottante pour être maintenu dans l'appel isolé. Lorsque l'appel isolé n'est pas intégré à une boucle, le temps système utilisé pour la sauvegarde et la restauration des registres supplémentaires peut neutraliser le gain acquis par la suppression des références mémoire.

Les fonctions spécifiées dans la liste des identificateurs sont autorisées pour examiner des objets externes et renvoyer des résultats dépendant de l'état de l'environnement d'exécution. Ces fonctions peuvent également modifier la mémoire pointée par un argument passé à la fonction, autrement dit, des appels par référence. Ne spécifiez aucune fonction qui s'auto-appelle ou repose sur une mémoire statique locale. Intégrer une telle fonction dans la directive **#pragma isolated_call** peut entraîner des résultats imprévisibles.

L'exemple suivant illustre l'utilisation de la directive **#pragma isolated_call**. Le compilateur suppose que l'appel de la fonction `this_function` ne changera pas la valeur de la variable externe `a` car cette fonction n'a pas d'effet secondaire. Dès lors, le compilateur peut supposer que l'argument de `other_function` a pour valeur 6.

```
int a, this_function(int);  /* Supposé sans effets secondaires */
#pragma isolated_call(this_function)
that_function()
{
   a = 6;
   this_function(7);  /* L'appel ne modifie pas la valeur de a */
   other_function(a);          /* L'argument "a" a la valeur 6 */
}
```

Niveaux d'optimisation

Les niveaux d'optimisation offerts par les compilateurs XL ont été modifiés par rapport aux versions antérieures. Les nouvelles options proposées sont :

Pas d'optimisation

En l'absence de toute forme d'indicateur **–O**, le compilateur génère un code "brut" sans restructuration des instructions ni autre forme d'amélioration.

–O ou –O2

Ces indicateurs (équivalents) demandent au compilateur d'effectuer une optimisation en se fondant sur une restructuration minimale du code. Seules sont appliquées les directives explicites de type **#pragma**. Ce niveau n'effectue plus de mise en pipeline logicielle, de déroulement de boucles ou de simple mise en commun prévisionnelle. Il définit également une contrainte sur la quantité de mémoire utilisable par le compilateur.

Il en résulte, par rapport aux versions antérieures, une possible dégradation des performances des routines étendues ou complexes compilées avec l'option **–O**.

–O3

Cette option impose au compilateur d'appliquer les techniques d'optimisation et d'utiliser le volume de mémoire nécessaire pour une optimisation maximale.

Ce niveau peut modifier les programmes au niveau fonctionnel si ces derniers tiennent compte :

- des exceptions à virgule flottante,
- du signe de zéro,
- des effets d'un réagencement des calculs.

Ces effets secondaires peuvent cependant être évités, au prix d'une diminution des performances, en combinant l'option **–qstrict** avec **–O3**.

L'option **–qhot** combinée à **–O3** permet d'effectuer des mises en commun prévisionnelles et quelques déroulements de boucle.

Par rapport à l'option **–O** des versions antérieures, ces modifications procurent le même niveau, voire un niveau plus élevé de performances des routines étendues ou complexes compilées avec l'option **–O3** (éventuellement combinée à **–qstrict** ou **–qhot**).

Options XL C pour l'optimisation de la sous-routine string.h

AIX permet d'imbriquer les sous-routines string dans le programme d'application au lieu de les utiliser à partir de **libc.a**. Cette opération économise le temps de la liaison appel/retour. Pour ce faire, le code source de l'application doit comporter l'instruction :

```
#include <string.h>
```

avant d'utiliser la (les) sous-routine(s). Sous la version 3.1, les seules sous-routines insérables par cette technique sont :

- **strcpy()**
- **strcmp()**

S'y ajoutent désormais les sous-routines :

- **strlen()**
- **strchr()**
- **strrchr()**
- **strcat()**
- **strncat()**

- **strncpy()**
- **strncmp()**
- **index()**
- **rindex()**
- **memchr()**
- **memcpy()**
- **memccpy()**
- **memmove()**
- **memcmp()**
- **memset()**

Pour revenir au niveau d'imbrication de la version 3.1, faites précéder l'instruction `#include <string.h>` de :

```
#define __STR31__
```

Style de codage optimisé C et C++

Dans un grand nombre de cas, le coût en termes de performances d'un module C n'est pas évident, voire contraire à ce qui était attendu. Voici quelques conseils à cet égard.

- Chaque fois que possible, utilisez le type `int` au lieu de `char` ou `short`.

  La plupart du temps, les types de données `char` et `short` requièrent davantage d'instructions. Le coût supplémentaire induit (en terme de temps) et, exception faite des grands tableaux, l'économie d'espace ainsi réalisée sont plus que neutralisés par l'accroissement de la taille de l'exécutable.

- Si vous devez utiliser le type `char`, déclarez-le `unsigned`, si possible.

  `signed char` exige deux instructions de plus que `unsigned char` à chaque chargement de la variable dans un registre.

- Optez de préférence pour des variables locales (automatiques) plutôt que des variables globales.

  L'accès aux variables globales demande plus d'instructions que l'accès aux variables locales. De plus, en l'absence d'informations, le compilateur suppose que toute variable globale a été modifiée par un appel de sous-routine. Ce phénomène a un effet inverse sur l'optimisation car la valeur d'une variable globale utilisée après un appel de sous-routine doit être rechargée.

- Chaque fois que vous devez accéder à une variable globale (non partagée par d'autres routines), copiez sa valeur dans une variable locale et utilisez cette copie.

  Sauf si la variable globale n'est utilisée qu'une fois, il est plus efficace d'utiliser la copie locale.

- Préférez les codes binaires aux chaînes pour les enregistrements et les tests.

  Les chaînes utilisent à la fois l'espace de données et l'espace d'instructions. Par exemple, la séquence :

```
#define situation_1 1
#define situation_2 2
#define situation_3 3
int situation_val;

situation_val = situation_2;
. . .
if (situation_val == situation_1)
. . .
```

est plus efficace que :

```
char situation_val[20];

strcpy(situation_val,”situation_2”);
. . .
if ((strcmp(situation_val,”situation_1”))==0)
. . .
```

- Lorsque les chaînes sont vraiment nécessaires, préférez les chaînes de longueur fixe aux chaînes de longueur variable terminée par zéro.

  Les routines de type **mem*()**, telles que **memcpy()**, sont plus rapides que les routines **str*()** correspondantes, telles que **strcpy()**, car les routines **str*()** doivent vérifier chaque octet à la recherche d’une valeur nulle, alors que les routines **mem*()** n’ont pas à le faire.

Temps d’exécution des compilateurs

Sous AIX, deux commandes permettent d’appeler le compilateur C : **cc** et **xlc**.
La commande **cc**, historiquement utilisée pour appeler le compilateur C du système lance le compilateur XL C en mode `langlevel=extended`. Ce mode permet de compiler des programmes C non conformes aux normes ANSI. Mais il consomme également du temps processeur.

Si le programme à compiler est conforme à ANSI, il est préférable d’appeler le compilateur XL C via la commande **xlc**.

Spécifier l’indicateur **–O3** englobe implicitement l’option **–qmaxmem**. Ainsi, le compilateur peut utiliser autant de mémoire que l’optimisation maximum l’exige. Il en résulte deux effets :

- Sur un système multi-utilisateur, une compilation **–O3** lourde peut utiliser une quantité de mémoire telle que les autres utilisateurs subiront une baisse des performances.
- Sur un système pauvre en mémoire réelle, une compilation **–O3** lourde peut utiliser une quantité de mémoire telle que le taux de pagination sera élevé et la compilation sérieusement ralentie.

Programmes à CPU limitée

Pour un programmeur sans cesse confronté aux limitations d’adressage imposées par certains environnements (tels que DOS), l’environnement du ESCALA avec ses segments de mémoire virtuelle de 256 Mo semble offrir des possibilités infinies. Il est alors tenté d’ignorer les contraintes de mémoire et de coder avec des longueurs de chemin minimales et une simplicité extrême. Cette ”négligence” n’est pas sans conséquences. La mémoire virtuelle est vaste, mais sa vitesse est variable. Plus la mémoire est utilisée, plus elle est ralentie selon une progression non linéaire. Tant que le volume total de mémoire virtuelle sollicitée par l’ensemble des programmes (c’est-à-dire le cumul des parties actives) est légèrement inférieur au volume de mémoire réelle non fixe, la mémoire virtuelle s’exécute à peu près à la même vitesse que la mémoire réelle. Dès que le volume sollicité dépasse le nombre de trames de page disponibles, les performances de la mémoire se dégradent (si la fonction de contrôle de charge est désactivée) selon une progression pouvant atteindre un facteur 2. On dit alors que le système *s’emballe* : il consacre presque tout son temps à la pagination, au détriment de tout travail productif, chaque processus tentant de reprendre aux autres la mémoire nécessaire pour alimenter sa partie active. Si la fonction de contrôle de charge mémoire VMM est active, cet auto-emballement perpétuel peut être évité, mais au prix d’un allongement considérable des temps de réponse.

Il est plus grave d’exploiter inefficacement la mémoire que les caches. En effet, l’écart de vitesse entre la mémoire et le disque est bien supérieur à celui entre la mémoire et le cache. Et si une absence en mémoire cache consomme quelques dizaines de cycles CPU, un défaut de page requiert plus de 20 millisecondes, soit au moins 400 000 cycles CPU.

Même si la présence de la fonction de contrôle de charge mémoire VMM sous AIX garantit qu'aucune condition d'emballement naissante ne dégénère en auto-emballement perpétuel, les temps de réponse et/ou le débit subissent l'effet des défauts de page inutiles.

Structuration du code paginable

Pour limiter la partie de code active d'un programme, regroupez le code fréquemment exécuté dans une zone réduite, séparée du code rarement exécuté. En particulier :

- Ne placez pas sur une ligne de longs blocs de code de traitement des erreurs. Insérez-les dans différentes sous-routines, de préférence dans des modules de code source distincts. Cette consigne s'applique non seulement aux chemins d'erreur, mais aussi à toute option fonctionnelle peu utilisée.
- Ne structurez pas les modules chargeables arbitrairement. Faites en sorte que les modules objet fréquemment appelés soient implantés le plus près possible des demandeurs. Il est préférable de concentrer en fin de module chargeable les modules objet composés (idéalement) de sous-routines peu appelées. Les page qu'ils renferment sont rarement lues.

Structuration des données paginables

Pour limiter la partie de données active, tentez de regrouper les données très sollicitées et évitez les références inutiles aux pages de mémoire virtuelle. En particulier :

- N'affectez via **malloc** ou **calloc** que l'espace nécessaire. Après **malloc**, n'initialisez jamais un tableau de taille maximale si, en réalité, seule une fraction de celui-ci est utilisée. Lorsque vous modifiez une nouvelle page pour initialiser les éléments du tableau, vous forcez VMM à voler une page de mémoire réelle. Cette opération entraînera un défaut de page lorsque le processus propriétaire de la page volée tentera d'y accéder. Pensez que la différence entre **malloc** et **calloc** ne se situe pas seulement au niveau de l'interface. Dans la mesure où **calloc** met à zéro l'espace mémoire affecté, il touche toutes les pages affectées, alors que **malloc** ne touche que la première page. Si vous affectez un espace étendu via **calloc** et n'utilisez au début qu'une petite portion de cet espace, vous chargez inutilement le système. Non seulement les pages doivent être initialisées, mais aussi, si leurs trames en mémoire réelle sont demandées, ces pages (qui ne seront jamais utilisées) doivent être écrites dans l'espace de pagination. Vous gaspillez ainsi des E/S et des emplacements de l'espace de pagination.
- Les mêmes problèmes peuvent se produire avec des listes liées de vastes structures (telles que des tampons). Si votre programme doit effectuer une série de chaînages à la recherche d'une clé, séparez liens et clés, d'une part, et données, d'autre part, ou optez pour un système de style table de hachage.
- Le regroupement référentiel doit être autant spatial que temporel : les structures de données doivent être initialisées juste avant leur utilisation (si tant est qu'elles le soient). Sur un système très chargé, si une longue période sépare l'initialisation de l'utilisation des structures de données, ces dernières risquent de se voir voler leurs trames. Le programme se heurte alors à un défaut de page dès qu'il tente d'utiliser la structure de données concernée.
- De même, si, une fois utilisée, une grande structure n'est plus sollicitée jusqu'à la fin du programme, elle doit être libérée. Mais, utiliser **free** pour libérer de l'espace affecté par **malloc** ou **calloc** ne suffit pas, car la commande **free** ne libère que les intervalles d'adresses occupés par la structure. Pour libérer de la mémoire réelle et de l'espace de pagination, vous devez renoncer à cet espace via la commande **disclaim**.

Mémoire fixe optimisée

Pour éviter les phénomènes de boucles et de temps d'attente, une petite fraction du système doit être fixée en mémoire réelle. La notion de partie active n'a alors plus de sens pour le code et les données impliquées puisque toutes les informations fixées se trouvent en permanence en mémoire réelle, qu'elles soient utilisées ou non. Les programmes (pilotes d'unité écrits par un utilisateur, par exemple) qui fixent un code ou une donnée

doivent être soigneusement élaborés (ou examinés, s'ils sont portés) pour s'assurer que les fractions de mémoire fixe sont utilisées avec parcimonie. Voici quelques conseils à ce sujet :

- Le code est fixé sur la base de module chargeable (fichier exécutable). Si certains modules objets d'un composant doivent être fixés et d'autres simplement paginés, veillez à placer les modules à fixer dans un module chargeable distinct.
- Il n'est pas souhaitable de fixer préventivement un module ou une structure de données : le concepteur doit savoir dans quelles conditions l'information peut être demandée et si, dans ce cas, un défaut de page peut ou non être toléré.
- Les structures fixes dont la taille dépend de la charge (telles que les pools de mémoire tampon) doivent être optimisables par l'administrateur système.

Conseils pour une installation performante

Cette section vous conseille sur les actions à exécuter avant et pendant l'installation.

Cette section traite des points suivants :

- "Préinstallation d'AIX"
- "Préinstallation de la CPU"
- "Préinstallation de la mémoire"
- "Préinstallation du disque"
- Préinstallation des cartes de communication : décrit à "Récapitulatif des paramètres d'optimisation d'UDP, de TCP/IP et de mbuf".

Préinstallation d'AIX

Installation d'AIX sur un nouveau système

Avant de commencer l'installation, vous devez avoir déterminé la taille et l'emplacement des systèmes de fichiers disque et des espaces de pagination, et le moyen de les communiquer à AIX.

Mise à niveau d'AIX sur un système existant

Pour installer une mise à niveau d'AIX :

- Identifiez, dans votre environnement actuel, toutes les utilisations des outils d'optimisation **schedtune** et **vmtune**, propres à cette version. Ces outils étant réservés à l'utilisateur `racine`, vous ne devriez pas avoir trop de mal à collecter ces informations.
- Si ces outils sont utilisés au cours de l'amorçage du système (à partir de **/etc/inittab**, par exemple), vous devez les désinstaller temporairement ou les désactiver jusqu'à être certain qu'ils fonctionnent correctement sur la nouvelle version.

Préinstallation de la CPU

Il est déconseillé de modifier *a priori* les paramètres de planification par défaut de la CPU (tels que la durée des tranches horaires). A moins que vous n'ayez une grande expérience en contrôle et optimisation d'une charge de travail similaire sur une configuration analogue, ne modifiez pas ces paramètres pendant l'installation.

Pour des indications sur la post-installation, reportez-vous à "Contrôle et optimisation de la CPU".

Préinstallation de la mémoire

Si vous installez un système de plus de 32 Mo, appelé à gérer plus de cinq utilisateurs simultanément, vous devez envisager d'augmenter le niveau minimum de multiprogrammation associé au mécanisme de contrôle de charge mémoire VMM. Par exemple, si vous estimez que, au bas mot, quatre des applications les plus gourmandes en mémoire doivent pouvoir s'exécuter simultanément, tout en laissant au moins 16 Mo pour le système d'exploitation et 25 % de mémoire réelle aux pages de fichiers, vous pouvez augmenter le niveau minimum de multiprogrammation de 2 (par défaut) à 4, par la commande :

```
# schedtune -m 4
```

Ne procédez à aucune autre modification de seuil de mémoire tant que vous n'avez pas confronté le système à la charge de travail réelle.

Pour des indications sur la post-installation, reportez-vous à "Contrôle et optimisation de la mémoire".

Préinstallation du disque

Recommandations générales

Bien que les mécanismes de définition et d'extension des volumes logiques adoptent les valeurs par défaut les plus satisfaisantes possibles, seul l'installateur du système est à même de déterminer la taille et l'emplacement des volumes logiques, en fonction du volume de données escompté et des contraintes de la charge de travail, pour des E/S optimisées. Dans cette optique, voici quelques recommandations utiles :

- Si possible, le groupe de volumes par défaut, **rootvg**, ne doit être constitué que du volume physique sur lequel le système a été initialement installé. Un ou plusieurs autres groupes de volumes doivent être définis pour contrôler les autres volumes physiques du système. Ces mesures sont utiles tant au niveau de la gestion que des performances.
- Pour améliorer les performances d'un groupe de volumes comportant plusieurs volumes physiques, vous pouvez :
  - Définir initialement le groupe de volumes avec un seul volume physique.
  - Définir un volume logique au sein du nouveau groupe de volume. Le volume logique journal du groupe de volumes est alors affecté sur le premier volume physique.
  - Ajouter les volumes physiques restants au groupe de volumes.
  - Définir les systèmes de fichiers très sollicités sur les nouveaux volumes physiques.
  - Définir exclusivement les systèmes de fichiers de très faible activité, s'il en existe, sur le volume physique contenant le volume logique du journal.

  Cette solution sépare les E/S de journalisation des E/S très importantes de données, augmentant la probabilité de chevauchement. Cette technique peut avoir un effet particulièrement marqué sur les performances du serveur NFS car l'écriture des données et du journal doit être achevée avant que NFS ne déclare les E/S terminées pour une opération d'écriture.
- Attribuez dès que possible aux volumes logiques leur taille maximale escomptée. Par précaution, traitez en premier les volumes logiques les plus importants en terme de performances pour être sûr qu'ils seront contigus et affectés à l'emplacement souhaité.
- Les volumes logiques très sollicités doivent occuper des portions de plusieurs unités de disque. Si l'option "**Gamme** de volumes physiques" du menu **smit** "Ajout volume logique" (raccourci **smit mklv**) est positionné sur **maximum**, le nouveau volume logique sera réparti sur les divers volumes physiques du groupe de volumes (ou du moins, le lot de volumes physiques explicitement spécifié).
- Si le système est équipé d'unités de différents types (ou que vous devez déterminer l'unité à commander), les remarques suivantes vous seront utiles :
  - Les gros fichiers normalement accessibles séquentiellement doivent être installés sur l'unité de disque la plus rapidement disponible. Les niveaux de performances en accès aléatoires et séquentiels mesurés sont, du plus lent au plus rapide :

Drive Capacity	SCSI Adapter	Random Pages per Second	Sequential Pages per Second
200MB	Model 250 Integrated	approx. 40	approx. 250
400MB	SCSI II	approx. 50	approx. 375
857MB	SCSI II	approx. 60	approx. 550
2.4GB	SCSI II	approx. 65*	approx. 525
1.37GB	SCSI II	approx. 70	approx. 800
540MB	SCSI II	approx. 85	approx. 975
1.0GB**	SCSI II	approx. 85	approx. 1075
2.0GB	SCSI II	approx. 85	approx. 950

* par accès (deux en tout)

** Cette unité de 1 Go (référence 45G9464) remplace l'ancienne unité de 1 Go (référence 55F5206) depuis fin 1993.

Remarque : Ces chiffres résultent de mesures effectuées en laboratoire dans des conditions idéales. Ils constituent la synthèse de différentes mesures, et non les résultat d'un seul essai comparatif. Nous vous les indiquons pour vous donner une idée générale des vitesses relatives des unités de disque. Ils sont appelé à évoluer avec les améliorations apportées aux unités, aux cartes et au logiciel.

– Si des accès séquentiels fréquents sont à prévoir sur les gros fichiers installés sur les unités de disques les plus rapides, il est conseillé de limiter le nombre de pilotes de disque par carte disque. Nous préconisons pour les unités de 540 Mo, 1 Go et 2 Go :

Disk Adapter	Disk Drives per Adapter
Original ESCALA SCSI adapter	1
SCSI-2 High Performance Controller	2
SCSI-2 Fast Adapter (8-bit)	2
SCSI-2 Fast/Wide Adapter (16-bit)	3

– Chaque fois que possible, raccordez les unités exigeant des performances très élevées à une carte SCSI-2. Ce type de carte offre des fonctionnalités, telles que l'écriture dos à dos, que les autres cartes disque du ESCALA ne proposent pas.

– Sur les unités de disque 200 Mo, 540 Mo et 1 Go, les volumes logiques appelés à recevoir des fichiers séquentiels fréquemment sollicités doivent être affectés sur le bord externe du volume physique. Ces disques ont davantage de blocs par piste dans leur section externe, ce qui améliore les performances des opérations séquentielles.

– Sur un bus SCSI, les unités dotées des adresses SCSI les plus élevées (telles que définies sur les unités physiques) sont prioritaires. Le plus souvent, cet effet est négligeable, mais les opérations séquentielles portant sur des fichiers volumineux sont connues pour exclure de l'accès au bus les unités dotées d'un petit numéro. Vous devrez probablement configurer les unités de disque contenant les données les plus critiques en matière de temps de réponse aux adresses les plus élevées de chaque bus SCSI. La commande **lsdev –Cs scsi** porte sur les affectations d'adresses actuelles sur chaque bus SCSI. Pour la carte SCSI d'origine, l'adresse SCSI est le premier nombre de la quadruple paire de nombres de la sortie. Dans l'exemple ci-dessus, le disque de 400 Mo réside à l'adresse SCSI 0, le disque de 320 Mo, à l'adresse 1, et l'unité de bande 8 mm, à l'adresse 5.

```
hdisk0   Available 00-01-00-00 400 MB SCSI Disk Drive
hdisk1   Available 00-01-00-10 320 MB SCSI Disk Drive
rmt0     Defined   00-01-00-50 2.3 GB 8mm Tape Drive
```

– Les gros fichiers très utilisés et généralement ouverts en mode aléatoire (tels que les bases de données) doivent être répartis sur plusieurs volumes physiques.

Pour des indications sur la post-installation, reportez-vous à ”Contrôle et optimisation des E/S disque”, page 8-1.

Position et taille des espaces de pagination

En règle générale, la taille cumulée des espaces de pagination doit être au moins double de celle de la mémoire réelle (256 Mo maximum, soit 512 Mo d’espace de pagination). Pour les mémoires supérieures à 256 Mo, la taille recommandée est :

```
espace de pagination total
= 512 Mo + (taille mémoire - 256 Mo) * 1,25
```

L’idéal est de disposer de plusieurs espaces de pagination de taille grosso modo équivalentes, implantés sur des unités de disque physiques distinctes. Si vous souhaitez ajouter des espaces de pagination, créez-les sur des volumes physiques moins chargés que le volume physique du groupe **rootvg**. VMM affecte alors quatre blocs de pagination en mode RR (répartition par permutation circulaire), issus des espaces de pagination actifs disposant d’espace. Lors de l’amorçage du système, seul l’espace de pagination principal (hd6) est actif. En conséquence, tous les blocs d’espace de pagination affectés pendant l’amorçage se trouvent sur l’espace de pagination principal. Ce qui signifie que l’espace de pagination principal doit être quelque peu plus vaste que les espaces de pagination secondaires. Pour un bon fonctionnement de l’algorithme de répartition par permutation circulaire, les espaces de pagination secondaires doivent être de même taille.

La commande **lsps –a** donne un instantané du niveau d’utilisation courant de tous les espaces de pagination d’un système. La sous-routine **psdanger()** peut également être utilisée pour déterminer les niveaux critiques d’utilisation de l’espace de pagination. Dans l’exemple ci-dessous, le programme utilise **psdanger()** pour afficher un message d’avertissement au-delà d’une certaine limite :

```
/* psmonitor.c
  Monitors system for paging space low conditions. When the condition is
  detected, writes a message to stderr.
  Usage:    psmonitor [Interval [Count]]
  Default:  psmonitor 1 1000000
*/
#include <stdio.h>
#include <signal.h>
main(int argc,char **argv)
{
  int interval = 1;         /* seconds */
  int count = 1000000;      /* intervals */
  int current;              /* interval */
  int last;                 /* check */
  int kill_offset;          /* returned by psdanger() */
  int danger_offset;        /* returned by psdanger() */
```

```
  /* are there any parameters at all? */
    if (argc > 1) {
    if ( (interval = atoi(argv[1])) < 1 ) {
      fprintf(stderr,"Usage: psmonitor [ interval [ count ] ]\n");
      exit(1);
    }
    if (argc > 2) {
      if ( (count = atoi( argv[2])) < 1 ) {
         fprintf(stderr,"Usage: psmonitor [ interval [ count ] ]\n");
         exit(1);
      }
    }
  }
  last = count -1;
  for(current = 0; current < count; current++) {
    kill_offset = psdanger(SIGKILL); /* check for out of paging space */
    if (kill_offset < 0)
      fprintf(stderr,
        "OUT OF PAGING SPACE! %d blocks beyond SIGKILL threshold.\n",
        kill_offset*(-1));
    else {
      danger_offset = psdanger(SIGDANGER); /* check for paging space low
*/
      if (danger_offset < 0) {
        fprintf(stderr,
          "WARNING: paging space low. %d blocks beyond SIGDANGER
threshold.\n",
          danger_offset*(-1));
        fprintf(stderr,
          "                              %d blocks below SIGKILL
threshold.\n",
          kill_offset);
      }
    }
      if (current < last)
        sleep(interval);
  }
}
```

Mise en miroir et performances

Si la fonction miroir est utilisée avec l'option Mirror Write Consistency activée (par défaut), vous devrez certainement installer les copies dans la région externe du disque, les informations de Mirror Write Consistency étant toujours inscrites sur le cylindre 0. La mise en miroir simple est coûteuse en terme de performances, elle l'est davantage encore assortie de l'option Write Verify (une rotation disque supplémentaire par écriture) et plus encore assortie des deux options Write Verify et Mirror Write Consistency (une rotation disque plus une recherche sur le cylindre 0). Pour éviter toute confusion, rappelons que bien qu'une commande **lslv** signale que l'option Mirror Write Consistency est activée pour les volumes logiques non miroir, aucun traitement n'est effectué si la valeur de COPIES n'est pas supérieure à 1. L'option Write Verify est, quant à elle, désactivée par défaut car elle n'a pas de signification (donc d'incidence) sur les volumes logiques non miroir.

Préinstallation des cartes de communication

Consultez le "Récapitulatif des paramètres d'optimisation d'UDP, TCP/IP et mbuf", page 9-31.

Chapitre 5. Contrôle système et diagnostics des performances

Ce chapitre décrit les outils et techniques de contrôle des performances du système. Les sections traitées sont les suivantes :

- ”Contrôle continu des performances”
- ”Commandes de contrôle **iostat**, **netstat** et **vmstat**”
- ”Utilitaire de diagnostic des performances”
- ”Diagnostics en fonction d’un symptôme particulier”
- ”Diagnostics à l’aide de PerfPMR”
- ”Identification des ressources limitatives”
- ”Gestion de la charge de travail”.

Contrôle continu des performances

Sur certaines installations, le contrôle des performances s'effectue à la demande : lorsqu'une anomalie de performances est relevée, l'analyste lance une ou plusieurs commandes pour déterminer l'origine de l'incident. Dans d'autres cas, il est nécessaire de recréer explicitement le problème pour collecter des données d'analyse. Le résultat est que les utilisateurs se heurtent deux fois au même problème de performance.

Il est souvent plus efficace d'exercer un contrôle en continu, de préférence avec relevé automatique de données supplémentaires en cas de dégradation des performances. Les avantages du contrôle continu en compensent largement le coût.

- En effet, un tel contrôle permet de : Détecter les anomalies naissantes avant qu'elles ne fassent sentir leurs effets.
- Détecter les incidents non signalés par les utilisateurs (réticents à effectuer cette démarche ou jugeant l'incident suffisamment mineur), mais qui affectent leur productivité et leur moral.
- Collecter les données générées dès la première apparition de l'incident.

Un contrôle efficace suppose les cinq opérations suivantes :

- Relevé périodique des informations de performances fournies par le système d'exploitation.
- Stockage des informations pour les diagnostics futurs.
- Communication des informations à l'administrateur système.
- Analyse plus approfondie des situations jugées critiques ou à la demande de l'administrateur système.
- Collecte et stockage des données détaillées utiles.

Les sections qui suivent présentent plusieurs méthodes de contrôle continu. Ces différentes approches ne sont pas incompatibles, mais les utiliser conjointement serait redondant.

- ”Commandes de contrôle **iostat**, **netstat** et **vmstat”**
- ”Utilitaire de diagnostic des performances”

Commandes de contrôle iostat, netstat et vmstat

Certaines caractéristiques fonctionnelles des commandes **iostat**, **netstat** et **vmstat** peuvent être avantageusement exploitées pour le contrôle continu des performances :

- Comptes rendus à un intervalle fixe sans limitation de durée.
- Rapports d'activité lors des variations liées à différents types de charge.
- Comptes rendus d'activité depuis le dernier rapport, facilitant la détection des changements intervenus.

Voici des extraits de comptes rendus périodiques générés par ces commandes :

```
$ iostat 5 2
tty:     tin       tout     cpu:   % user    % sys     % idle    % iowait
         0.0        0.0              0.0       0.2       99.6       0.1

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.1       0.3       0.0      18129     56842
cd0              0.0       0.0       0.0          0         0

tty:     tin       tout     cpu:   % user    % sys     % idle    % iowait
         0.0        0.0              23.1      9.0       65.9       2.0

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           2.4       6.4       1.6          0        32
cd0              0.0       0.0       0.0          0         0


$ vmstat 5 2
procs    memory              page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0  2610  1128   0   0   0   0    0   0 112    1  19  0  0 99  0
 0  0  2505  1247   0   0   0   0    0   0 125 1056  37 22  9 67  2


$ netstat -I tr0 5
   input    (tr0)     output           input   (Total)    output
 packets  errs  packets  errs colls   packets  errs  packets  errs colls
  532099  1664      985     0     0    532111  1664      997     0     0
      45     0        6     0     0        45     0        6     0     0
      44     1        5     0     0        44     1        5     0     0
```

Le premier compte rendu généré par chacune de ces commandes concerne l'activité cumulée depuis le dernier amorçage du système. Le second compte rendu montre l'activité enregistrée durant le premier intervalle de 5 secondes. De ce simple extrait, il ressort que l'activité enregistrée au cours du premier intervalle a été sensiblement supérieure à la moyenne.

Ces commandes sont à la base du mécanisme de contrôle. Les scripts shell peuvent être écrits pour réduire les données générées par la commande ***stat**, signaler les problèmes de performances ou enregistrer les données concernant l'état du système au moment de l'incident. Par exemple, un script shell peut tester le pourcentage de CPU disponible, et lorsqu'il détecte un pourcentage égal à 0 signifiant que la CPU est saturée, exécuter un autre script shell. Par exemple :

```
$ ps -ef | egrep -v "STIME|$LOGNAME" | sort +3 -r | head -n 15
```

Ce script enregistre les 15 processus actifs (autres que les processus de l'utilisateur du script) qui ont récemment consommé le plus de temps CPU.

Selon le niveau de sophistication requis, élaborer un ensemble de scripts shell peut s'avérer difficile. Fort heureusement, il existe des modules moins exigeants en développement et configuration qui offrent bien plus de fonctions que la plupart des installations n'en requièrent localement.

Chapitre 6. Contrôle et optimisation de la CPU

L'unité centrale d'un ESCALA est l'un des composants les plus rapides du système : il est rarissime qu'un programme monopolise 100 % de la CPU au-delà de quelques secondes. Même sur des systèmes multiutilisateurs très chargés, il arrive qu'au bout d'une période de 10 ms tous les travaux soient en attente. Si un message indique que la CPU est occupée à 100 % (0 % disponible et 0 % d'attente) depuis un certain temps, il est fort probable qu'un programme se trouve engagé dans une boucle infinie. Dans ce cas, si le programme en cause est simplement coûteux, mais tout à fait exploitable, il convient de l'identifier et de lui appliquer un traitement ad hoc.

Ce chapitre indique comment détecter les programmes incontrôlés, ou faisant un usage intensif de CPU, et réduire leur impact sur les performances.

Si vous êtes peu familiarisé avec la programmation de la CPU sous AIX, reportez-vous à "Performances du programmateur CPU sous AIX", page 2-2, avant de poursuivre.

Cette section traite des points suivants :

- "Contrôle de la CPU via **vmstat"**
- "Mesure de la CPU via **time**"
- "Identification des grands consommateurs de CPU via **ps**"
- "Analyse des programmes via **tprof**"
- "Analyse du flux de commande via **stem**"
- "Restructuration des exécutables via **fdpr**"
- "Contrôle des conflits pour la CPU"
- "Modification de la tranche horaire du programmateur"
- "Administration de l'ID utilisateur par rapport à la CPU"

L'outil **xmperf** est également très utile pour contrôler l'utilisation de la CPU.

Contrôle de la CPU via vmstat

Comme contrôleur de CPU, **vmstat** est plus pratique à utiliser que **iostat**. Chaque relevé est présenté sur une ligne distincte, ce qui en facilite la lecture. **vmstat** propose également des informations d'ordre général sur l'utilisation de la mémoire, la pagination et les E/S disque ordinaires. L'exemple qui suit indique comment interpréter la sortie, et notamment identifier les conditions d'emballement d'un programme ou de carence en CPU en environnement multiutilisateur.

```
$ vmstat 2
procs    memory              page                  faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  0 22478  1677   0   0   0   0    0   0 188 1380 157 57 32  0 10
 1  0 22506  1609   0   0   0   0    0   0 214 1476 186 48 37  0 16
 0  0 22498  1582   0   0   0   0    0   0 248 1470 226 55 36  0  9

 2  0 22534  1465   0   0   0   0    0   0 238  903 239 77 23  0  0
 2  0 22534  1445   0   0   0   0    0   0 209 1142 205 72 28  0  0
 2  0 22534  1426   0   0   0   0    0   0 189 1220 212 74 26  0  0
 3  0 22534  1410   0   0   0   0    0   0 255 1704 268 70 30  0  0
 2  1 22557  1365   0   0   0   0    0   0 383  977 216 72 28  0  0

 2  0 22541  1356   0   0   0   0    0   0 237 1418 209 63 33  0  4
 1  0 22524  1350   0   0   0   0    0   0 241 1348 179 52 32  0 16
 1  0 22546  1293   0   0   0   0    0   0 217 1473 180 51 35  0 14
```

Cette sortie met en évidence l'impact d'un programme en boucle dans un environnement multiutilisateur. Il ressort des trois premiers relevés (le résumé ayant été supprimé) que l'équilibre du système pour l'utilisation de la CPU se situe à 50–55 % côté utilisateur, à 30–35 % côté système et à 10–15 % en attente d'E/S. Dès le lancement du programme en boucle, tous les cycles CPU disponibles sont utilisés. Aucune E/S n'étant effectuée dans une boucle, tous les cycles non utilisés du fait de l'attente des E/S sont alors exploités par le programme. En fait, ce processus est capable de s'approprier la CPU de tout processus utile qui s'en déferait. Bénéficiant du même niveau de priorité que les autres processus d'avant-plan, le processus n'est pas contraint d'abandonner de la CPU au profit d'un autre processus devenu diffusable. Son exécution dure environ 10 secondes (5 relevés). Au-delà, l'activité du système suit un cours plus normal.

Mesure de la CPU via time

La commande **time** est un outil simple mais efficace qui permet de mieux comprendre les caractéristiques de performances d'un programme. **time** indique le temps écoulé entre le début et la fin du programme (rubrique `real`). Elle indique également le temps CPU utilisé par le programme. Le temps CPU est divisé entre temps `user` et temps `sys`. La valeur `user` correspond au temps utilisé par le programme lui-même et, le cas échéant, par les sous-routines de bibliothèque qu'il appelle. La valeur `sys` correspond au temps utilisé par les appels système effectués par le programme (directement ou indirectement).

La somme des valeurs `user` + `sys` correspond au coût CPU direct total de l'exécution du programme. Ce coût n'inclut pas les coûts CPU imputables aux portions du noyau exécutées pour le programme en dehors de sa routine. Par exemple, au lancement du programme, les trames de page retirées de la liste des disponibilités doivent être remplacées. La récupération de trames de page à cet effet représente un coût qui n'est pas englobé dans la consommation CPU du programme.

La différence entre le temps CPU `real` et le temps CPU total :

```
real – (user + sys)
```

correspond à la somme de tous les facteurs susceptibles de retarder le programme, plus les coûts indéterminés propres au programme. Ces facteurs sont, par ordre d'importance décroissante :

- les E/S requises pour charger le texte et les données du programme,
- les E/S requises pour acquérir la mémoire réelle nécessaire à l'exploitation du programme,
- le temps CPU consommé par d'autres programmes,
- le temps CPU consommé par le système d'exploitation.

Dans l'exemple qui suit, le programme présenté dans la section précédente a été compilé avec l'option –O3 pour le rendre plus rapide. L'écart entre le temps réel d'exécution du programme et le temps CPU total (utilisateur et système) est minime. Le programme bénéficie de tout le temps nécessaire, probablement au détriment d'autres programmes.

```
$ time looper
real    0m3.58s
user    0m3.16s
sys     0m0.04s
```

Dans l'exemple suivant, le programme est exécuté avec une priorité moindre (valeur nice augmentée de 10). La durée de son exécution est multipliée par 2, mais les autres programmes ont l'opportunité d'effectuer leurs travaux :

```
$ time nice -10 looper
real    0m6.54s
user    0m3.17s
sys     0m0.03s
```

Notez que la commande **nice** a été insérée dans la commande **time**, et non l'inverse. Si vous entrez :

```
$ nice -10 time looper
```

c'est la version **/usr/bin/time** de la commande **time** qui est utilisée et non la nouvelle version intégrée à **ksh** – qui génère un relevé plus détaillé. Il en est de même si vous appelez **time** via une autre commande. Si la commande **time** est exécutée en premier, vous obtenez la version intégrée, sauf si vous spécifiez le nom qualifié complet de **/usr/bin/time**. Si **time** est appelé à partir d'une autre commande, vous obtenez **/usr/bin/time**.

Conseils relatifs à time et timex

Vous devez tenir compte de certaines contraintes lorsque vous utilisez **time** ou sa variante **timex** :

- L'utilisation des commandes /**usr**/**bin**/**time** et /**usr**/**bin**/**timex** est déconseillée. Chaque fois que possible, optez pour la sous-commande **time** du shell Korn ou C. « Sous la version 3.2.5, le temps CPU comptabilisé par /**usr**/**bin**/**time** est incorrect lorsqu'il est lié à l'exploitation d'un script shell renfermant des séquences de commandes chaînées par des tubes : seule la dernière commande de la séquence est comptabilisée dans le temps CPU, le reste est perdu. Cette erreur s'explique par le fait que /**usr**/**bin**/**time** utilise le shell par défaut du système. Or le shell par défaut de la version 3.2.5 est le shell Bourne, lequel exécute (commande **exec**) les séquences de commandes de telle sorte que seule la consommation CPU de la dernière commande peut être mesurée. En revanche, le shell Korn (shell par défaut d'AIX version 4.1) ne présente pas le même inconvénient.
- La commande **timex –s** passe par **sar** pour obtenir des statistiques supplémentaires. **sar** étant intrusive, **timex –s** l'est également. Les données reportées par **timex –s**, notamment dans le cas d'exécutions brèves, risquent de ne pas refléter fidèlement le comportement d'un programme exécuté sur un système non contrôlé.
- La durée d'une impulsion d'horloge (10 millisecondes) et les règles suivies par le programmateur pour attribuer du temps CPU aux routines faussent légèrement le résultat de la commande **time**. Des variations se produisent immanquablement entre les exécutions successives. Ces variations sont définies en termes d'implulsions d'horloge. Naturellement, ces variations sembleront d'autant plus importante que le temps d'exécution du programme est court.
- Il est déconseillé d'utiliser la commande **time** ou **timex** (à partir de /**usr**/**bin** ou par le biais de la fonction **time** du shell intégré) pour mesurer le temps système ou utilisateur consommé par une séquence de commandes chaînées par des tubes et entrées sur la ligne de commande. En effet, si la commande **time** n'est pas spécifiée correctement, elle peut ne mesurer qu'une seule commande de la séquence. Il faut savoir par ailleurs que l'erreur n'est pas signalée, la syntaxe étant techniquement correcte.

Contrôle de la CPU via xmperf

Le recours à **xmperf** pour afficher l'utilisation de la CPU dans le système est encore plus flagrant. Si vous affichez la CPU sous la forme d'un graphique représentant une ligne d'horizon en mouvement, et la CPU de l'utilisateur en rouge vif, vous pourrez immédiatement constater depuis l'autre bout de la pièce l'emballement d'un programme. La commande **xmperf** est décrite en détail dans le manuel *Performance Toolbox 1.2 and 2.1 for AIX: User's Guide*.

Identification des gros consommateurs de CPU via ps

Le compte rendu fourni par la commande **ps** peut être présenté dans trois colonnes différentes :

Colonne	Valeur
C	Temps CPU récemment utilisé pour le processus.
TIME	Temps CPU total utilisé par le processus depuis son lancement.
%CPU	Temps CPU total utilisé par le processus depuis son lancement divisé par le temps écoulé depuis le lancement du processus. Mesure la dépendance du programme vis-à-vis de la CPU.

Le script shell :

```
$ps -ef | egrep -v 'STIME|$LOGNAME' | sort +3 -r | head -n 15
```

sert à mettre en évidence les processus utilisateur du système les plus gourmands en CPU. Appliqué à l'exemple de la section "Contrôle de la CPU via vmstat", ce script générerait la sortie suivante (la ligne d'en-tête a été ajoutée pour faciliter la lecture) :

```
    USER   PID  PPID   C    STIME    TTY  TIME CMD
  waters 45742 54701 120 15:19:05 pts/29  0:02 ./looper
    root 52121     1  11 15:32:33 pts/31 58:39 xhogger
    root  4250     1   3 15:32:33 pts/31 26:03 xmconsole allcon
    root 38812  4250   1 15:32:34 pts/31  8:58 xmconstats 0 3 30
    root 27036  6864   1 15:18:35      -  0:00 rlogind
    root 47418 25925   0 17:04:26      -  0:00 coelogin <d29dbms:0>
    bick 37651 43538   0 16:58:40  pts/4  0:00 /bin/ksh
    bick 43538     1   0 16:58:38      -  0:07 aixterm
     luc 60061 27036   0 15:18:35 pts/18  0:00 -ksh
```

L'utilisation récente de la CPU est donnée à la quatrième colonne ("C"). Le processus du programme en boucle est en tête de la liste. Notez que la valeur fournie est probablement sous-estimée car au-delà de 120 le programmateur s'arrête de compter.

ps est un outil très souple qui permet d'identifier les programmes en cours sur le système et les ressources qu'ils mobilisent. Les options et colonnes associées sont décrites dans la documentation de référence de **ps**, le manuel *AIX Commands Reference*.

Les indicateurs de **ps** relèvent pour la plupart d'une des catégories suivantes :

1. Indicateurs spécifiant les processus à inclure dans la sortie.
2. Indicateurs spécifiant les informations à afficher pour chaque processus.

Les tableaux ci-après récapitulent les différents indicateurs de la commande **ps** et leurs effets. Dans les deux tableaux, les indicateurs spécifiés avec un signe moins sont situés à gauche et les autres, à droite. Les deux groupes d'indicateur s'excluent mutuellement : si le premier est spécifié avec un signe moins, tous les autres indicateurs de la commande **ps** doivent obligatoirement appartenir au groupe des indicateurs dotés du signe moins.

			Indicateurs de spécification de processus :										
					–G				–U				
Les processus répertoriés sont :	**–A**	**–a**	**–d**	**–e**	**–g**	**–k**	**–p**	**–t**	**–u**	**a**	**g**	**t**	**x**
Tous les processus	**Y**	–	–	–	–	–	–	–	–	–	**Y**	–	–
Non leaders du groupe de processus et non associés à un terminal	–	**Y**	–	–	–	–	–	–	–	–	–	–	–
Non leaders du groupe de processus	–	–	**Y**	–	–	–	–	–	–	–	–	–	–
Processus non noyau	–	–	–	**Y**	–	–	–	–	–	–	–	–	–
Membres des groupes de processus spécifiés	–	–	–	–	**Y**	–	–	–	–	–	–	–	–
Processus noyau	–	–	–	–	–	**Y**	–	–	–	–	–	–	–
Ceux spécifiés dans la liste des numéros de processus	–	–	–	–	–	–	**Y**	–	–	–	–	–	–
Ceux associés avec les TTY dans la liste	–	–	–	–	–	–	**– Y** (nTTYs)		–	**– –** (1 TTY)		**Y**	–
Processus utilisateur spécifiés	–	–	–	–	–	–	–	–	**Y**	–	–	–	–
Processus avec terminaux	–	–	–	–	–	–	–	–	–	**Y**	–	–	–
NON associés à un TTY	–	–	–	–	–	–	–	–	–	–	–	–	**Y**

Spécifiée sans indicateur, la commande **ps** s'applique par défaut à tous les processus de l'émetteur de cette commande **ps**.

		Indicateurs de sélection de colonne								
				–U						
Colonne :	**Default1**	**–f**	**–l**	**–u**	**Default2**	**e**	**l**	**s**	**u**	**v**
PID	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
TTY	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
TIME	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
CMD	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
USER	**–**	**Y**	**–**	**–**	**–**	**–**	**–**	**–**	**Y**	**–**

UID	–	–	Y	Y	–	–	Y	–	–	–
PPID	–	Y	Y	–	–	–	Y	–	–	–
C	–	Y	Y	–	–	–	Y	–	–	–
STIME	–	Y	–	–	–	–	–	–	Y	–
F	–	–	Y	–	–	–	–	–	–	–
S/STAT	–	–	Y	–	Y	Y	Y	Y	Y	Y
PRI	–	–	Y	–	–	–	Y	–	–	–
NI/NICE	–	–	Y	–	–	–	Y	–	–	–
ADDR	–	–	Y	–	–	–	Y	–	–	–
SZ/SIZE	–	–	Y	–	–	–	Y	–	Y	Y
WCHAN	–	–	Y	–	–	–	Y	–	–	–
RSS	–	–	–	–	–	–	Y	–	Y	Y
SSIZ	–	–	–	–	–	–	–	Y	–	–
%CPU	–	–	–	–	–	–	–	–	Y	Y
%MEM	–	–	–	–	–	–	–	–	Y	Y
PGIN	–	–	–	–	–	–	–	–	–	Y
LIM	–	–	–	–	–	–	–	–	–	Y
TSIZ	–	–	–	–	–	–	–	–	–	Y
TRS	–	–	–	–	–	–	–	–	–	Y
environment (suivant la commande ; sans en-tête de colonne)	–	–	–	–	Y	–	–	–	–	

Si **ps** est lancée sans indicateurs ou avec un indicateur de spécification de processus commençant par un signe moins, les colonnes s'affichent comme celles illustrées pour Default1. Si la commande est assortie d'un indicateur de spécification de processus ne commençant pas par un signe moins, les colonnes correspondant à Default2 sont affichées. **–u** ou **–U** est à la fois un indicateur de processus et de sélection de colonne.

Voici une brève description du contenu des colonnes :

PID	ID processus
TTY	Terminal ou pseudo-terminal associé au processus.
TIME	Cumul du temps CPU consommé, en minutes et secondes.
CMD	Commande exécutée par la processus.
USER	Nom de connexion du propriétaire du processus.
UID	ID utilisateur numérique du propriétaire du processus.
PPID	ID du processus parent.
C	Temps CPU récemment utilisé.
STIME	Heure de début du processus (s'il a été lancé le jour même). Sinon, date de début du processus.
F	Valeur hexadécimale de 8 caractères décrivant les indicateurs associés au processus (voir la description détaillée de la commande **ps**).
S/STAT	Etat du processus (voir la description détaillée de la commande **ps**).
PRI	Niveau de priorité courant du processus.

NI/NICE	Valeur nice du processus.
ADDR	Numéro de segment de pile de processus.
SZ/SIZE	Nombre de pages de segments actifs touchées, multiplié par 4.
WCHAN	Événement attendu par le processus.
RSS	Cumul du nombre de pages de segments actifs et du nombre de pages de segments de codes en mémoire, multiplié par 4.
SSIZ	Taille de la pile du noyau.
%CPU	Taux d'utilisation de la CPU par le processus depuis son lancement.
%MEM	Valeur RSS divisée par la taille de la machine en ko, multipliée par 100 et arrondie au pourcentage le plus proche.
PGIN	Nombre de chargements de page consécutifs à un défaut de page. Équivaut approximativement au volume des E/S, toutes les E/S étant comptabilisées sous AIX comme des défauts de page.
LIM	Limite sur la taille RSS. Affiché "xx" si non définie.
TSIZ	Taille de la section de texte du fichier exécutable.
TRS	Nombre de pages de segments de code, multiplié par 4.
environnement	Valeur de toutes les variables d'environnement du processus.

Analyse des programmes via tprof

L'exécution d'un programme est une combinaison variable de codes d'application, de bibliothèques, de sous-routines et de services de noyau. Généralement, un programme non optimisé utilise la plupart des cycles CPU qui lui sont impartis à un nombre réduit d'instructions ou de sous-routines. Ces zones critiques sont souvent inattendues pour la personne chargée de mettre le programme en application et peuvent être considérées comme des anomalies de performances. L'outil préconisé pour localiser les zones critiques d'un programme est le "profileur" **tprof**. **tprof**, fondé sur la fonction de suivi, est capable de profiler un programme (analyser son exécution) généré par un des compilateurs XL suivants : XL C, XL C++, XL FORTRAN.

Sous AIX version 4.1, le programme **tprof** est intégré à la boîte à outils PTX (Performance Toolbox for AIX). Pour savoir si **tprof** est disponible, entrez :

```
lslpp -lI perfagent.tools
```

Si ce module est installé, **tprof** est disponible.

Les données brutes de **tprof** sont obtenues par la fonction de suivi (reportez-vous au chapitre "Analyse des performances avec l'utilitaire de suivi", page 11-1). Lorsqu'un programme est profilé, l'utilitaire de suivi est activé et chargé de recueillir les données du point d'ancrage du suivi (ID 234) qui enregistre le contenu du registre d'adresse d'instruction (IAR) à chaque interruption d'horloge système (100 fois par seconde). Plusieurs autres points d'ancrage sont également activés pour permettre à **tprof** de suivre le processus et de diffuser l'activité. Les enregistrements de suivi ne sont pas inscrits dans un fichier disque mais vers un tube. Un programme lit le contenu du tube et construit une table avec les adresses de programme uniques rencontrées et leur nombre d'occurrences. Cette table est copiée sur le disque lorsque toute la charge de travail a été traitée. Le composant chargé de la réduction des données dans **tprof** fait la corrélation entre les adresses d'instruction rencontrées et les intervalles d'adresses occupés par les programmes. Puis, il diffuse les occurrences d'adresses ("impulsions d'horloge") parmi les programmes impliqués dans la charge.

La diffusion des impulsions d'horloge est globalement proportionnelle au temps CPU utilisé sur chaque programme (10 millisecondes par impulsion). Lorsque les programmes fortement sollicités ont été identifiés, le programmeur peut prendre les mesures qui s'imposent pour restructurer les zones critiques ou minimiser leur usage.

Exemple théorique

Le programme C suivant initialise à 0x01 chaque octet d'un grand tableau de nombres entiers, incrémente chaque entier (`int`) d'une constante aléatoire et envoie en sortie un entier (`int`) désigné aléatoirement. Le programme ne remplit pas de fonction particulière, mais il est représentatif des programmes qui traitent de grands tableaux.

```
/*  Array Incrementer -- Version 1  */
#include <stdlib.h>
#define Asize 1024
#define RowDim InnerIndex
#define ColDim OuterIndex
main()
{
  int Increment;
  int OuterIndex;
  int InnerIndex;
  int big [Asize][Asize];
  /* initialize every byte of the array to 0x01 */
  for(OuterIndex=0; OuterIndex<Asize; OuterIndex++)
  {
    for (InnerIndex=0; InnerIndex<Asize; InnerIndex++)
      big[RowDim][ColDim] = 0x01010101;
  }
  Increment = rand();
  /* increment every element in the array */
  for(OuterIndex=0; OuterIndex<Asize; OuterIndex++)
  {
    for (InnerIndex=0; InnerIndex<Asize; InnerIndex++)
    {
      big[RowDim][ColDim] += Increment;
      if (big[RowDim][ColDim] < 0)
       printf(”Negative number. %d\n”,big[RowDim][ColDim]);
    }
  }
  printf(”Version 1 Check Num: %d\n”,
        big[rand()%Asize][rand()%Asize]);
  return(0);
}
```

Ce programme a été compilé par la commande :

```
$ xlc -g version1.c -o version1
```

Le paramètre `-g` commande au compilateur XL C de générer le module objet avec des informations de mise au point symboliques destinées à **tprof**. Bien que **tprof** soit capable de profiler des modules optimisés, nous avons omis de spécifier le paramètre `-O` pour préciser les numéros de ligne utilisés par **tprof**. Lors de l'optimisation, le compilateur XL C réorganise généralement les codes si bien que l'interprétation de la sortie de **tprof** est plus difficile. Sur le système test, ce programme mobilise environ 5,97 secondes, dont plus de 5,9 secondes de temps CPU utilisateur. Le programme remplit donc parfaitement son objectif : être limité en temps CPU.

Appliquer au programme la commande **tprof** comme suit :

```
$ tprof -p version1 -x version1
```

génère un fichier `__version1.all` (voir ci-dessous) qui indique le nombre d'impulsions CPU consommées par chaque programme impliqué dans l'exécution.

```
    Process      PID    Total   Kernel     User   Shared    Other
    =======      ===    =====   ======     ====   ======    =====
   version1 33570480      793       30      763        0        0
        bsh 33566383        8        8        0        0        0
 /etc/init        1        6        0        6        0        0
/etc/syncd     3059        5        5        0        0        0
     tprof     5038        4        2        2        0        0
   rlogind    11345        2        2        0        0        0
   PID.771      771        1        1        0        0        0
     tprof    11940        1        1        0        0        0
     tprof    11951        1        1        0        0        0
     tprof    13987        1        1        0        0        0
       bsh    16048        1        1        0        0        0
   =======      ===    =====   ======     ====   ======    =====
     Total              823       52      771        0        0

    Process     FREQ    Total   Kernel     User   Shared    Other
    =======      ===    =====   ======     ====   ======    =====
   version1        1      793       30      763        0        0
        bsh        2        9        9        0        0        0
 /etc/init        1        6        0        6        0        0
/etc/syncd        1        5        5        0        0        0
     tprof        4        7        5        2        0        0
   rlogind        1        2        2        0        0        0
   PID.771        1        1        1        0        0        0
   =======      ===    =====   ======     ====   ======    =====
     Total       11      823       52      771        0        0

Total Ticks For version1(    USER) =    763

       Subroutine   Ticks     %           Source   Address  Bytes
    =============  ======  ======       =======   =======  =====
            .main     763    92.7      version1.c       632    560
```

La première section du relevé indique le nombre d'impulsions consommées par chaque processus (ou qui lui sont imputables). Le programme `version1` a utilisé 763 impulsions parmi lesquelles 30 se sont déroulées dans le noyau mais sont imputées à `version1`. Deux processus exécutant le shell Bourne sont impliqués dans l'exécution de `version1`. Quatre processus ont exécuté des codes liés à **tprof**. Le processus **init**, le démon **sync**, un processus **rlogin** et un autre processus représentent 14 impulsions.

Rappelons que le programme associé à un ID processus numérique donné change à chaque appel **exec**. Si un programme d'application en exécute (**exec**) un autre, le nom des deux programmes apparaît dans la sortie de **tprof** associée au même ID processus.

La seconde section du rapport récapitule les résultats par programme, indépendamment de l'ID processus. Elle indique le nombre (`FREQ`) de processus distincts susceptibles d'atteindre chaque programme à un point donné.

La troisième section analyse les impulsions d'horloge utilisateur associées au programme exécutable profilé. L'analyse indique le nombre d'impulsions utilisées par chaque fonction de l'exécutable et le pourcentage que ce nombre représente sur le total des impulsions CPU produites lors de l'exécution (`823`).

Jusque là, aucune opération **tprof** n'a fait appel à la version compilée spéciale du programme : la même analyse aurait pu être effectuée sur un programme dont le code source est inaccessible.

Il ressort clairement de ce relevé que le programme lui-même est le plus grand consommateur de temps CPU (92,7 %), et non le noyau ou les sous-routines de bibliothèques du programme. Etudions-le de plus près.

La compilation de `version1.c` ayant été faite avec l'option **–g**, le fichier objet comprend des informations qui associent les décalages du texte du programme aux lignes du code source. Par conséquent, `tprof` a créé une version annotée du fichier source `version1.c`, appelée `__t.version1.c`, basée sur les décalages et les informations relatives aux numéros de ligne dans le module objet. La première colonne reporte le numéro de ligne.

La seconde comptabilise le nombre de fois où le point d'ancrage a signalé les interruptions de l'horloge alors que le système exécutait l'une des instructions associées à cette ligne.

```
Ticks Profile for main in version1.c

   Line   Ticks   Source

    14     34       for(OuterIndex=0; OuterIndex<Asize; OuterIndex++)
    15      -       {
    16     40         for (InnerIndex=0; InnerIndex<Asize; InnerIndex++)
    17    261           big[RowDim][ColDim] = 0x01010101;
    18      -       }
    19      -       Increment = rand();
    20      -
    21      -       /* increment every element in the array */
    22     70       for(OuterIndex=0; OuterIndex<Asize; OuterIndex++)
    23      -       {
    24      -         for (InnerIndex=0; InnerIndex<Asize; InnerIndex++)
    25      -         {
    26     69          big[RowDim][ColDim] += Increment;
    27     50          if (big[RowDim][ColDim] < 0)
    28    239            printf("Negative number.%d\n",
                                 big[RowDim][ColDim]);
    29      -         }
    30      -       }
    31      -       printf("Version 1 Check Num: %d\n",
    32      -                    big[rand()%Asize][rand()%Asize]);
    33      -       return(0);
    34      -     }

  763 Total Ticks for main in version1.c
```

Ce compte rendu montre clairement que le plus grand nombre d'impulsions est associé à l'accès aux éléments du tableau `big`, et nous permet d'optimiser les performances en nous concentrant sur les boucles internes `for`. La première boucle `for` (initialisation) est un exemple de programmation sauvage : il est totalement inefficace d'initialiser les éléments des tableaux les uns après les autres. S'il s'agissait de mettre le tableau à 0, nous aurions dû utiliser **bzero**. Dans la mesure où il s'agit de donner à chaque bit la valeur d'un caractère spécifique, nous allons utiliser **memset** pour remplacer la première boucle `for`.
(les fonctions **bzero** et **memset** sont très efficaces, tout comme les fonctions **str** : écrites en assembleur, elles exploitent des instructions matérielles qui n'ont pas d'équivalent direct en C).

Nous devons accéder successivement aux éléments du tableau pour les incrémenter, mais en nous assurant que le schéma de la référence mémoire se rapporte à des adresses consécutives pour optimiser l'utilisation du cache. Ici, les dimensions des rangées sont modifiées plus rapidement que celles des colonnes. Les tableaux C étant structurés sur la base des rangées, nous omettons une rangée complète à chaque référence mémoire.
La longueur des rangées étant de 1 024 entiers – `int` – (4 096 octets), nous changeons de page à chaque référence. La taille du tableau dépassant largement les possibilités du cache de données et du TLB de données, nous avons écrit un programme qui accepte le maximum de cache et de TLB. Pour ce faire, nous avons simplement transposé les instructions `#define` pour inverser les valeurs de `RowDim` et `ColDim`.

La forme non optimisée de ce programme (`version2.c`) consomme environ 2,7 secondes de temps CPU, à comparer aux 7,9 secondes pour `version1`.

Le fichier suivant, `__t.version2.c`, résulte d'une commande **tprof** appliquée à la forme non optimisée du programme :

```
Ticks Profile for main in version2.c

   Line   Ticks   Source

    15      -       memset(big,0x01,sizeof(big));
    16      -       Increment = rand();
    17      -
    18      -       /* increment in memory order */
    19     60       for(OuterIndex=0; OuterIndex<Asize; OuterIndex++)
    20      -       {
    21      -         for (InnerIndex=0; InnerIndex<Asize; InnerIndex++)
    22      -         {
    23     67          big[RowDim][ColDim] += Increment;
    24     60          if (big[RowDim][ColDim] < 0)
    25     43          printf(”Negative number.
%d\n”,big[RowDim][ColDim]);
    26      -         }
    27      -       }
    28      -       printf(”Version 2 Check Num: %d\n”,
    29      -                   big[rand()%Asize][rand()%Asize]);
    30      -       return(0);
    31      -     }

   230 Total Ticks for main in version2.c
```

En étant attentif au schéma d'utilisation du temps CPU, nous avons multiplié la vitesse CPU du programme – non optimisé – par trois. Les performances sont multipliées par 7 si `version1.c` et `version2.c` sont optimisés avant compilation.

L'utilisation de la CPU par un programme intervient principalement dans les sous-routines de bibliothèque qu'il utilise et non dans le programme lui-même. Si, dans `version2.c`, le test conditionnel de la ligne 24 et l'entrée `printf` de la ligne 28 sont supprimés pour créer la `version3.c` suivante :

```
#include <string.h>
#include <stdlib.h>
#define Asize 256
#define RowDim OuterIndex
#define ColDim InnerIndex

main()
{
  int Increment;
  int OuterIndex;
  int InnerIndex;
  int big [Asize][Asize];

  /* Initialize every byte to 0x01 */
  memset(big,0x01,sizeof(big));
  Increment = rand();
  /* increment in memory order */
  for(OuterIndex=0; OuterIndex<Asize; OuterIndex++)
  {
    for (InnerIndex=0; InnerIndex<Asize; InnerIndex++)
    {
      big[RowDim][ColDim] += Increment;
      printf(”RowDim=%d, ColDim=%d, Number=%d\n”,
              RowDim, ColDim, big[RowDim][ColDim]);
    }
  }
  return(0);
}
```

le temps d'exécution est essentiellement imputable à l'instruction **printf**. La commande :

```
$ tprof -v -s -k -p version3 -x version3 >/dev/null
```

génère une __version3.all qui comprend les données de profilage pour le noyau et la bibliothèque de sous-routine partagée libc.a (seule bibliothèque utilisée par ce programme) :

```
    Process      PID    Total   Kernel     User   Shared    Other
    =======      ===    =====   ======     ====   ======    =====
   version3 33568373      818       30       19      769        0
        bsh 33567348        5        5        0        0        0
      tprof    15987        3        1        2        0        0
      tprof     7784        1        1        0        0        0
      tprof    12905        1        1        0        0        0
        bsh    13941        1        1        0        0        0
    =======      ===    =====   ======     ====   ======    =====
      Total              829       39       21      769        0

    Process     FREQ    Total   Kernel     User   Shared    Other
    =======     ===     =====   ======     ====   ======    =====
   version3       1      818       30       19      769        0
        bsh       2        6        6        0        0        0
      tprof       3        5        3        2        0        0
    =======     ===     =====   ======     ====   ======    =====
      Total       6      829       39       21      769        0

Total Ticks For version3(    USER) =     19

       Subroutine   Ticks     %           Source   Address  Bytes
    =============  ======  ======        =======   =======  =====
            .main      11     1.3      version3.c       632    320
          .printf       8     1.0         glink.s      1112     36

Total Ticks For version3(  KERNEL) =     30

       Subroutine   Ticks     %           Source   Address  Bytes
    =============  ======  ======        =======   =======  =====
         .sc_flih       7     0.8          low.s     13832   1244
        .i_enable       5     0.6          low.s     21760    256
        .vmcopyin       3     0.4        vmmove.c    414280    668
       .xix_setattr       2     0.2     xix_sattr.c    819368    672
       .isreadonly       2     0.2        disubs.c    689016     60
            .lockl       2     0.2         lockl.s     29300    208
         .v_pagein       1     0.1    v_getsubs1.c    372288   1044
         .curtime        1     0.1         clock.s     27656     76
         .trchook        1     0.1          noname     48168    856
           .vmvcs        1     0.1          vmvcs.s     29744   2304
       .spec_rdwr        1     0.1    spec_vnops.c    629596    240
            .rdwr        1     0.1         rdwr.c     658460    492
           .imark        1     0.1         isubs.c    672024    184
           .nodev        1     0.1      devsw_pin.c    135864     32
       .ld_findfp        1     0.1       ld_libld.c    736084    240

Total Ticks For version3( SH-LIBs) =    769

    Shared Object   Ticks     %           Source   Address  Bytes
    =============  ======  ======        =======   =======  =====
     libc.a/shr.o     769    92.0         /usr/lib    794624 724772

Profile: /usr/lib/libc.a  shr.o

Total Ticks For version3(/usr/lib/libc.a) =    769

       Subroutine   Ticks     %           Source   Address  Bytes
    =============  ======  ======        =======   =======  =====
          ._doprnt     476    56.9        doprnt.c     36616   7052
           .fwrite     205    24.5        fwrite.c     50748    744
           .strchr      41     4.9        strchr.s     31896    196
           .printf      18     2.2        printf.c    313796    144
          ._moveeq      16     1.9        memcmp.s     36192    184
           .strlen      10     1.2      strerror.c     46800    124
           .isatty       1     0.1        isatty.c     62932    112
          ._xwrite       1     0.1        flsbuf.c      4240    280
          .__ioctl       1     0.1         ioctl.c     57576    240
```

Ce qui confirme que les impulsions sont majoritairement imputables aux bibliothèques partagées— `libc.a`, ici. Le profil de `libc.a` indique que la plupart de ces impulsions sont imputables à la sous-routine **_doprnt**.

_doprnt est le module de traitement de **printf**, **sprintf**, etc. Avec une simple modification, nous avons augmenté le temps de traitement de 2,7à 8,6 secondes, et notre impression formatée consomme à présent environ 60 % du temps CPU. Ceci explique pourquoi le formatage doit être employé judicieusement. **Les performances de _doprnt** sont également affectées par l'environnement local. Reportez–vous à l'annexe I, "Support NLS local et vitesse". Ces tests ont été effectués dans l'environnement local C, le plus efficace.

Analyse du flux de commande via stem

Le module d'instrumentation **stem** peut suivre le flux de commande grâce à une gamme variée de logiciels. Sous AIX version 4.1, cet outil fait partie de la boîte à outils Performance Toolbox for AIX. Pour savoir si **stem** est disponible, entrez :

```
lslpp -lI perfagent.tools
```

Si ce module est installé, **stem** est disponible.

Les avantages les plus significatifs de **stem** sont :

- **stem** peut instrumenter des programmes d'application :
  - limités,
  - optimisés,
  - en multitraitement,
  - dans des bibliothèques partagées non limitées.
- les sous–routines d'instrumentation d'entrée et de sortie **stem** peuvent être :
  - fournies par **stem**,
  - fournies par l'utilisateur.

stem crée des versions instrumentées des programmes et des bibliothèques requises, et les stocke dans le répertoire /**tmp**/**EXE**. Lorsque l'utilisateur exploite ces programmes instrumentés, **stem** crée un fichier correspondant appelé **stem_out**.

Analyse de stem

Pour analyser le flux de commande d'un simple programme d'application, entrez :

```
stem -p stem_test_pgm
```

La sortie de cette commande est :

```
*******************************************************
Make.Stem does not exist, issueing make for stem_samples.o
make stem_samples.o
Target stem_samples.o is up to date.
*******************************************************
The instrumented stem_test_pgm is at /tmp/EXE/stem_test_pgm
Assuming version 3.2.5 or later, SVC_string=.sc_flih
```

L'instrumentation de `stem_test_pgm` a abouti, bien que le programme ait été limité. La forme instrumentée du programme réside dans le répertoire /**tmp**/**EXE**. Entrez ensuite :

```
/tmp/EXE/stem_test_pgm
```

Vous obtenez un fichier appelé **stem_out** dans le répertoire de travail courant. Dans ce cas, **stem_out** contient :

```
  Seconds.usecs  TID  Routine Names & Seconds.usecs since
entering routine.
767549539.847704   1 ->main
767549539.880523   1           ->setPI
767549539.880958   1           <-setPI  0.000435
767549539.881244   1           ->squareit
767549539.881515   1           <-squareit  0.000271
767549539.881793   1           ->printf
767549539.883316   1           <-printf  0.001523
767549539.883671   1           ->setPI
767549539.883944   1           <-setPI  0.000273
767549539.884221   1           ->squareit
767549539.884494   1           <-squareit  0.000273
767549539.884772   1           ->printf
767549539.885981   1           <-printf  0.001209
767549539.886330   1 <-main  0.038626
767549539.886647   1           ->exit
```

Le graphique d'appel capture les appels dirigés vers les fonctions du module (setPI et squareit) et vers la sous–routine **printf** de **libc.a**. Les numéros à droite des noms des sous–routines représentent le temps écoulé (en secondes et microsecondes) entre l'appel et la réponse.

Si le même processus est exécuté sur la commande **wc** (/**usr**/**bin**/**wc**), le fichier **stem_out** contient alors (pour **wc** d'un fichier composé de deux mots) :

```
  Seconds.usecs  TID  Routine Names & Seconds.usecs since
entering routine.
767548812.962031   1 ->main
767548812.993952   1           ->setlocale
767548812.995065   1           <-setlocale  0.001113
767548812.995337   1           ->catopen
767548812.995554   1           <-catopen  0.000217
767548812.995762   1           ->getopt
767548812.996101   1           <-getopt  0.000339
767548812.996345   1           ->open
767548812.996709   1           <-open  0.000364
767548812.996953   1           ->read
767548812.997209   1           <-read  0.000256
767548812.997417   1           ->read
767548812.997654   1           <-read  0.000237
767548812.997859   1           ->wcp
767548812.998113   1           ->printf
767548812.998586   1           <-printf  0.000473
767548812.998834   1           <-wcp  0.000975
767548812.999041   1           ->printf
767548813.000439   1           <-printf  0.001398
767548813.000720   1           ->close
767548813.000993   1           <-close  0.000273
767548813.001284   1           ->exit
```

Ce graphique d'appel, obtenu assez facilement, illustre la structure d'une commande AIX. Les appels à destination de **setlocale** et **catopen** assurent que la commande se déroule dans le même environnement local NLS (National Language Support) et avec le même catalogue de messages que son processus père.

Bien que les programmes **stem** puissent s'exécuter dans plusieurs processus, le graphique d'appel indique uniquement l'ordre d'exécution dans le processus principal.

Restructuration des exécutables via fdpr

Le programme **fdpr** (feedback-directed program restructuring) optimise des modules exécutables pour accélérer leur exécution et rendre plus efficace l'utilisation de la mémoire réelle. Sous AIX version 4.1, cet outil fait partie de la boîte à outils Performance Toolbox for AIX. Pour savoir si **fdpr** est disponible, entrez :

```
lslpp -lI perfagent.tools
```

Si ce module est installé, **fdpr** est disponible.

Le traitement **fdpr** se fait en trois étapes :

- Instrumentation du module exécutable à optimiser pour permettre la collecte détaillée des données de fonctionnement.
- Exploitation de l'exécutable instrumenté, avec une charge de travail fournie par l'utilisateur, et enregistrement des données de performance recueillies.
- Exploitation des données de performances pour piloter le processus d'optimisation, qui se traduit par un module exécutable restructuré et capable d'exécuter plus efficacement *la charge de travail gérée par l'exécutable instrumenté*. Il est crucial que la charge de travail utilisée pour piloter **fdpr** corresponde précisément à l'utilisation actuelle du programme. Les performances d'un exécutable restructuré avec des charges de travail différentes de celles utilisées pour piloter **fdpr** sont imprévisibles, mais peuvent être moins bonnes que celles de l'exécutable d'origine.

Par exemple, la commande :

```
fdpr -p NomProgramme -R3 -x test.sh
```

utilise le jeu d'essai `test.sh` pour exploiter le programme instrumenté `NomProgramme`. La sortie de cette commande peut être utilisée pour atteindre le plus haut degré d'optimisation (R3) du programme et former un nouveau module appelé, par défaut, `NomProgramme.fdpr`. La différence de performance entre l'exécutable optimisé et celui non-optimisé dépend largement de la précision de la simulation de la charge de travail par l'essai `test.sh`.

> **Attention :** Les algorithmes d'optimisation avancés intégrés au programme **fdpr** entraînent parfois une différence de fonctionnement entre les exécutables optimisés et le module exécutable d'origine. Pour une question de fiabilité, il est **essentiel** de tester de façon exhaustive un exécutable optimisé avant toute exploitation.

En résumé, les utilisateurs de **fdpr** doivent :

- veiller à utiliser une charge de travail représentative pour piloter **fdpr** ;
- tester de façon exhaustive le fonctionnement de l'exécutable restructuré ;
- exploiter l'exécutable restructuré uniquement sur les charges de travail pour lesquelles il est prévu.

Contrôle des conflits pour la CPU

Priorité des processus utilisateur

Les priorités des processus utilisateur peuvent être fixées via les commandes **nice** et **renice** et la sous-routine **setpri**, et affichées via **ps**. Les priorités sont présentées dans ”Priorité des processus et des routines”, page 2-3.

Exécution d’une commande à priorité non standard via nice

Tout utilisateur peut exploiter une commande à un niveau de priorité inférieur au niveau normal via **nice** . En revanche, seul l’utilisateur `root` peut lancer **nice** pour exploiter des commandes à une priorité plus élevée.

Via **nice**, l’utilisateur spécifie une valeur à ajouter ou à soustraire de la valeur nice standard. La valeur ainsi obtenue est exploitée par le processus qui lance la commande. A ce stade, la priorité du processus n’est toujours pas fixée : sa valeur est recalculée périodiquement en fonction de l’utilisation de la CPU, de la valeur nice ainsi que de la valeur minimum de la priorité du processus utilisateur.

La valeur nice standard d’un processus d’avant-plan est de 20, celle d’un processus d’arrière-plan, de 24. Pour obtenir la valeur initiale de la priorité, additionnez la valeur nice et le niveau minimum de priorité de processus utilisateur (40). Par exemple, la commande :

```
$ nice -5 vmstat 10 3 >vmstat.out
```

exécute **vmstat** avec la valeur nice de 25 (au lieu de 20), entraînant une priorité de base de 65 (avant ajout dû à l’utilisation récente de la CPU).

Un utilisateur `root` aurait pu lancer **vmstat** à une priorité supérieure :

```
# nice --5 vmstat 10 3 >vmstat.out
```

Si un utilisateur non `root` lance cette commande **nice**, la commande **vmstat** fonctionne, mais avec la valeur nice standard 20, et **nice** ne génère aucun message d’erreur.

Définition d’une priorité fixe via setpri

Une application qui fonctionne sous l’ID utilisateur `root` peut exploiter la sous-routine **setpri** pour définir sa propre priorité ou celle d’un autre processus. Voici quelques conseils :

```
retcode = setpri(0,59);
```

affecte au processus en cours une priorité fixe de 59. Si **setpri** échoue, il retourne la valeur –1.

Le programme suivant accepte une valeur de priorité et une liste d’ID de processus ; la priorité de tous les processus est fixée à la valeur indiquée.

```
/*
   fixprocpri.c
   Usage: fixprocpri priority PID . . .
*/

#include <sys/sched.h>
#include <stdio.h>
#include <sys/errno.h>

main(int argc,char **argv)
{
   pid_t ProcessID;
   int Priority,ReturnP;

   if( argc < 3 ) {
      printf(" usage - setpri priority pid(s) \n");
      exit(1);
   }

   argv++;
   Priority=atoi(*argv++);
   if ( Priority < 50 ) {
   printf(" Priority must be >= 50 \n");
      exit(1);
   }

   while (*argv) {
      ProcessID=atoi(*argv++);
      ReturnP = setpri(ProcessID, Priority);
   if ( ReturnP > 0 )
          printf("pid=%d new pri=%d  old pri=%d\n",
            (int)ProcessID,Priority,ReturnP);
      else {
          perror(" setpri failed ");
            exit(1);
      }
   }
}
```

Affichage de la priorité des processus via ps

L'indicateur –l (L minuscule) de la commande **ps** permet l'affichage des valeurs nice et des priorités courantes pour les processus spécifiés. Par exemple, pour afficher les priorités des processus d'un utilisateur donné, entrez :

```
# ps -lu waters
      F S UID  PID PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  241801 S 200 7032 7287   0  60 20 1b4c   108          pts/2  0:00 ksh
  200801 S 200 7569 7032   0  65 25 2310    88  5910a58  pts/2  0:00 vmstat
  241801 S 200 8544 6495   0  60 20 154b   108          pts/0  0:00 ksh
```

La sortie montre le résultat de la commande **nice –5** décrite plus haut. Le processus 7569 a une priorité effective de 65 (la commande **ps** a été lancée en mode superutilisateur lors d'une autre session, d'où la présence de deux TTY).

Si l'un des processus a exploité la sous-routine **setpri** pour s'attribuer une priorité fixe, le format en sortie de **ps –l** est le suivant :

```
     F S UID  PID PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
200903 S   0 10759 10500 0  59 -- 3438  40  4f91f98  pts/0  0:00 fixpri
```

Modification de la priorité d'un processus en cours via renice

Remarque : Dans cette section, la syntaxe **renice** pour AIX version 3 est utilisée. La section suivante traite de la syntaxe de **nice** et **renice** sous AIX versions 3 et 4.

renice modifie la valeur nice et, par conséquent, la priorité du ou des processus en cours d'exécution. Les processus sont identifiés via l'ID processus, l'ID groupe de processus ou le nom de l'utilisateur des processus. **renice** ne peut être appliqué à des processus à priorité fixe.

A partir de l'exemple précédent, modifions la valeur nice (**renice**) du processus **vmstat** lancé via **nice** :

```
# renice -5 7569
7569: old priority -5, new priority 0
# ps -lu waters
      F S UID  PID PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  241801 S 200 7032 7287   0  60 20 1b4c   108           pts/2  0:00 ksh
  200801 S 200 7569 7032   0  55 15 2310    92  5910a58  pts/2  0:00 vmstat
  241801 S 200 8544 6495   0  60 20 154b   108           pts/0  0:00 ksh
```

Le processus est à présent exécuté à une priorité *supérieure* à celles des autres processus d'avant-plan. Notez que **renice** n'ajoute ni ne retranche la valeur spécifiée à l'ancienne valeur nice. Elle remplace l'ancienne valeur par la nouvelle. Pour annuler le tout, entrez :

```
# renice -0 7569
7569: old priority -5, new priority 0
# ps -lu waters
      F S UID  PID PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  241801 S 200 7032 7287   0  60 20 1b4c   108           pts/2  0:00 ksh
  200801 S 200 7569 7032   1  60 20 2310    92  5910a58  pts/2  0:00 vmstat
  241801 S 200 8544 6495   0  60 20 154b   108           pts/0  0:00 ksh
```

Dans ces exemples, **renice** est lancé par l'utilisateur `root`. L'utilisation de **renice** par un utilisateur non root est doublement limitée. Il ne peut :

- lancer cette commande que sur ses propres processus,
- augmenter la priorité d'un processus, même pour restaurer sa priorité par défaut, après l'avoir diminuée via **renice**.

Clarification de la syntaxe nice/renice

AIX version 3

Pour les commandes **nice** et **renice**, l'ajout d'une valeur à la valeur nice standard (20) n'est pas exprimée de la même façon.

Pour **nice**, un signe moins placé avant la valeur, supposée positive, est obligatoire pour l'identifier. Pour spécifier une valeur négative, un second signe moins (sans espace entre les deux) est requis.

Pour **renice**, le paramètre qui suit le nom de la commande est supposé être la valeur à ajouter. Cette valeur peut comporter un signe ou non (dans ce cas, la valeur est positive). Les commandes suivantes sont donc équivalentes :

		Resulting nice Value	**Resulting Priority Value**
nice -5	**renice 5**	25	65
nice -5	**renice +5**	25	65
nice - -5	**renice -5**	15	55

AIX version 4.1

Sous AIX version 4, la syntaxe de **renice** a été modifiée pour compléter la syntaxe de **nice**, qui utilise l'indicateur **–n** pour identifier l'augmentation de la valeur nice. Le tableau suivant est la version AIX version 4.1 du tableau précédent :

		Resulting nice Value	**Resulting Priority Value**
nice -n 5	**renice -n 5**	25	65
nice -n +5	**renice -n +5**	25	65
nice -n -5	**renice -n -5**	15	55

Optimisation du calcul de la priorité d'un processus via schedtune

Une récente amélioration de **schedtune** et du programmateur CPU d'AIX autorise la modification des paramètres utilisés pour calculer la priorité d'un processus.
Cette nouveauté d'AIX version 4.1 est disponible dans le PTF pour la version 3.2.5.
Reportez-vous à "Priorité des processus et des routines", page 2-3.

La formule de calcul d'une priorité est la suivante :

```
priorité = priorité de base + valeur nice + (pénalité CPU basée
sur l'utilisation récente de la CPU)
```

La valeur de l'utilisation récente de la CPU d'un processus donné est incrémentée de 1 à chaque fois que le processus contrôle la CPU lors d'une interruption de l'horloge (toutes les 10 millisecondes). Cette valeur est affichée en colonne "C" de la sortie de la commande **ps**. Son maximum est 120.

L'algorithme calcule la pénalité CPU en divisant l'utilisation récente de la CPU par 2. Le *rapport pénalité CPU/utilisation récente de la CPU* est donc 0,5. Cette valeur est désignée par *R*.

Toutes les secondes, l'algorithme en cours divise la valeur de l'utilisation récente de la CPU de chaque processus par 2. Le *facteur de décroissance du temps d'utilisation de la CPU récente* est donc de 0,5. Cette valeur est désignée par *D*.

Pour certains utilisateurs, l'algorithme existant n'établit pas une distinction assez nette entre les processus d'arrière-plan et d'avant-plan. Par exemple, si un processus d'avant-plan (valeur nice = 20) et un processus d'arrière-plan (valeur nice = 24) lancés en même temps sollicitent de façon intensive les fonctions de calcul du système, on obtient la séquence suivante (sans tenir compte des autres activités) :

- Le processus d'avant-plan est programmé le premier. Au bout de 8 tranches horaires (80 ms), sa pénalité CPU est 4, ce qui rend sa priorité égale à celle du processus d'arrière-plan. L'algorithme de répartition par permutation circulaire engendre la programmation du processus d'arrière-plan.
- Au bout de 2 autres tranches horaires, la pénalité CPU du processus d'arrière-plan est 1, ce qui rend sa priorité supérieure de 1 à celle du processus d'avant-plan. Le processus d'avant-plan est programmé.
- Au bout de 2 autres tranches horaires, les priorités des processus redeviennent égales. Les processus continuent d'alterner toutes les 2 tranches horaires jusqu'à la fin de la seconde.
- A la fin de la seconde, le processus d'avant-plan aura disposé de 54 tranches horaires et celui d'arrière-plan de 46. Lorsque le facteur de décroissance est appliqué, les valeurs du temps d'utilisation CPU sont 27 et 23. A la deuxième seconde de leur compétition, les processus d'avant-plan n'ont que 4 tranches horaires de plus que le processus d'arrière-plan.

Même si le processus d'arrière-plan est lancé avec **nice –20**, la différence entre avant-plan et arrière-plan est à peine plus marquée. Bien que le programmateur arrête de compter les tranches horaire après 120, cela permet au niveau de pénalité CPU de s'arrêter à 60, ce qui suffit largement pour compenser la différence maximale de 40 de valeur nice.

Pour assouplir le processus d'attribution des priorités, les nouvelles caractéristiques permettent à l'utilisateur d'optimiser le rapport pénalité CPU/utilisation de CPU récente (*R*) et le taux de décroissance du temps d'utilisation de CPU récente (*D*). Cette optimisation est réalisée via deux nouvelles options de la commande **schedtune** : **–r** et **–d**. Chaque option spécifie un paramètre (entier compris entre 0 et 32) qui multiplie l'utilisation de CPU récente par la valeur du paramètre puis divise le résultat par 32 (en décalant de 5 à droite). Les valeurs **r** et **d** par défaut sont 16, ce qui entraîne le même résultat que l'algorithme de départ (*D*=*R*=16/32=0,5). Les nouvelles valeurs autorisent plus de latitude. Par exemple :

```
# schedtune -r 0
```

(*R*=0, *D*=0,5) signifie que la pénalité CPU est 0, ce qui rend la priorité absolue : aucun processus d'arrière-plan ne peut obtenir de temps CPU tant qu'un processus d'avant-plan en réclame. Les priorités des processus sont en effet constantes, même si, techniquement, ce ne sont pas des processus à priorité fixe.

```
# schedtune -r 5
```

(*R*=0,15625, *D*=0,5) signifie que des processus d'avant-plan n'entreront jamais en compétition avec un processus d'arrière-plan lancé via **nice –20**. La limite de 120 tranches de temps CPU accumulées signifie que la pénalité maximum pour les processus d'avant-plan est fixée à 18.

```
# schedtune -r 6 -d 16
```

(*R*=0,1875, *D*=0,5) signifie qu'un processus d'arrière-plan lancé via **nice –20**, le sera au moins une seconde avant que le processus d'arrière-plan ne reçoive du temps CPU. Cependant, les processus d'avant-plan peuvent toujours être distingués sur la base de l'utilisation de CPU. Les processus d'avant-plan longs qui doivent normalement se trouver en arrière-plan finissent par accumuler suffisamment d'utilisation de CPU pour les empêcher d'interférer avec l'avant-plan réel.

```
# schedtune -r 32 -d 32
```

(*R*=1, *D*=1) signifie que les processus longs atteignent une valeur C égale à 120 et la conservent, en opposition avec leur valeur nice. Les nouveaux processus ont la priorité quelle que soit leur valeur nice, jusqu'à ce qu'ils aient accumulé suffisamment de tranches horaires pour passer dans la gamme de priorités des processus existants.

Si vous estimez qu'un ou deux paramètres doivent être modifiés pour s'adapter à la charge de travail, entrez la commande **schedtune**, en tant qu'utilisateur `root`. Les valeurs modifiées seront conservées jusqu'à la prochaine commande **schedtune** ou jusqu'au réamorçage suivant du système. Vous pouvez rétablir les valeurs par défaut via **schedtune –D**, mais n'oubliez pas qu'alors *tous* les paramètres **schedtune** sont réinitialisés par cette commande, y compris les paramètres de contrôle de charge mémoire VMM. Pour faire perdurer une modification au fil des réamorçages, ajoutez la ligne appropriée à la fin du fichier **/etc/inittab**.

Modification de la tranche horaire du programmateur

Vous pouvez modifier la durée de la tranche horaire du programmateur via la commande **schedtune** (voir page A-6). La syntaxe de cette fonction est la suivante :

```
schedtune –t increase
```

increase est le nombre d'impulsions d'horloge de 10 ms à ajouter à la tranche horaire standard (un battement de 10 ms). Par exemple, **schedtune –t 2** fait passer la durée de la tranche horaire à 30 ms. **schedtune –t 0** lui réaffecte sa valeur par défaut.

Dans un environnement où la tranche horaire a été augmentée, certaines applications n'ont pas besoin d'utiliser la totalité de la tranche horaire. Ces applications peuvent renoncer explicitement au processeur via l'appel système **yield** (comme peuvent le faire les programmes dans un environnement non modifié). Après un appel **yield**, la routine qui émet l'appel est déplacée à la fin de la file d'attente de répartition en fonction de son niveau de priorité.

Administration des ID utilisateur

Pour optimiser le temps de réponse de la connexion et conserver du temps CPU dans un système multiutilisateur, AIX peut exploiter une version à accès direct du fichier **/etc/passwd** pour consulter les ID utilisateur. Lorsque cette fonction est utilisée, le fichier **/etc/passwd** existe toujours mais n'est pas employé dans le traitement courant.
Les versions à accès direct du fichier (**/etc/passwd.dir** et **/etc/passwd.pag**) sont créées via la commande **mkpasswd**. Si ces versions à accès direct ne sont pas actives, le processus de connexion revient à une recherche séquentielle lente, utilisant la CPU de façon intensive via **/etc/passwd**.

Lorsque les fichiers de mot de passe à accès direct ont été créés, et que les commandes **passwd**, **mkuser**, **chuser** et **rmuser** (ou leurs équivalents **smit**) sont exploitées pour administrer les ID utilisateur, les fichiers à accès direct sont mis à jour automatiquement.
Si le fichier **/etc/passwd** est modifié via un éditeur ou la commande **pwdadm**, les fichiers à accès direct doivent être reconstruits via la commande suivante :

```
# mkpasswd /etc/passwd
```

Chapitre 7. Contrôle et optimisation de la mémoire

La mémoire d'un ESCALA est presque toujours pleine. Si les programmes en cours ne l'occupent pas intégralement, AIX y conserve les pages de texte de programmes antérieurs et des fichiers associés. Ceci ne coûte rien, puisque la mémoire est de toutes façons inutilisée. Souvent, par contre, les pages de fichier ou de programme sont réutilisées, ce qui réduit les E/S disque.

Cette technique de mise en mémoire cache améliore l'efficacité du système mais rend plus difficile l'évaluation des besoins mémoire réels d'une charge de travail.

Ce chapitre décrit comment mesurer et modifier la quantité de mémoire utilisée. Elle comporte les sections suivantes :

- "Quantité de mémoire utilisée" répertorie les commandes relatives à l'exploitation de la mémoire et traite de leurs avantages et inconvénients.
- "Programme à perte de mémoire" décrit l'une des causes les plus courantes de surcharge de mémoire.
- "Analyse de l'exploitation de la mémoire via BigFoot" décrit les caractéristiques et le fonctionnement de l'outil BigFoot, qui rend compte des schémas d'utilisation de la mémoire.
- "Estimation de la mémoire requise via **rmss**" illustre les techniques d'utilisation de l'outil Reduced-Memory System Simulator.
- "Optimisation du contrôle de la charge mémoire VMM" décrit les situations dans lesquelles cette optimisation est nécessaire et les méthodes d'optimisation via la commande **schedtune**.
- "Optimisation du remplacement de page VMM" détaille les méthodes et les effets de la modification des paliers de remplacement de page VMM.

Les lecteurs qui ne seraient pas familiarisés avec la gestion de la mémoire virtuelle d'AIX peuvent se reporter à "Performance du gestionnaire de mémoire virtuelle (VMM)", page 2-5 avant de poursuivre.

Quantité de mémoire utilisée

Plusieurs outils rendent compte de l'utilisation de la mémoire. Les relevés les plus intéressants sont ceux établis via **vmstat**, **ps** et **svmon**.

vmstat

vmstat rend compte de la mémoire virtuelle "active" totale exploitée par tous les processus du système ainsi que le nombre de trames de page en mémoire réelle dans la liste des disponibilités. La mémoire virtuelle active est définie comme le nombre de pages de segments actifs en mémoire virtuelle qui ont été exploitées. Elle est généralement égale au nombre d'emplacements d'espace de pagination déjà affectés. Ce nombre peut être supérieur au nombre de trames de page réelles de la machine, dans la mesure où certaines pages de la mémoire virtuelle active peuvent avoir été évacuées vers l'espace de pagination.

ps

ps fournit des comptes rendus différents en fonction de l'indicateur spécifié. Le plus complet est issu de **ps v**, qui affiche les colonnes suivantes :

SIZE	Taille virtuelle en kilo-octets de la section de données du processus. (affiché comme SZ par les autres indicateurs). Ce chiffre est égal à la quantité de pages de segment de travail des processus en cause (le nombre d'emplacements d'espaces de pagination affectés) multiplié par 4. Si certaines pages de segment de travail sont évacuées, ce chiffre est supérieur à la quantité de mémoire réelle utilisée.
RSS	Taille de la mémoire réelle du processus (jeu résidant) en kilo-octets. Ce chiffre est égal à la somme de segments de travail et des pages de segments de code en mémoire multipliée par 4. Les pages de segment de codes sont partagées entre toutes les instances en cours du programme. Si 26 processus **ksh** sont en cours, une seule copie d'une page donnée de l'exécutable **ksh** réside en mémoire, mais **ps** intègre la taille de ce segment de code dans le RSS de *chaque* instance de **ksh**.
TSIZ	Taille de l'image texte (programme partagé). Taille de la section de texte du fichier exécutable. Les pages de texte de l'exécutable ne sont chargées en mémoire que lorsqu'elles sont utilisées (branchement ou chargement). Ce chiffre ne représente que la limite supérieure de la quantité de texte qui peut être chargée.
TRS	Taille de l'ensemble de texte résidant (en mémoire réelle). Elle est égale au nombre de pages de segment de codes multiplié par 4. Comme indiqué plus haut, ce chiffre surestime l'utilisation de la mémoire pour les programmes dont plusieurs instances sont en cours.
%MEM	Somme du nombre de segments de travail et de pages de segments de code en mémoire multipliée par 4 (c'est-à-dire la valeur RSS), divisée par la taille de la mémoire réelle de la machine en ko, multipliée par 100, et arrondie. Cette valeur est une estimation du pourcentage de mémoire réelle utilisée par le processus. Malheureusement, comme RSS, le coût d'un processus partageant un programme avec d'autres y est souvent surévalué. De plus, ce pourcentage étant arrondi, tous les processus du système ayant une valeur RSS inférieure à 0,005 fois la taille de la mémoire réelle ont une valeur %MEM égale à 0.

Il ressort clairement qu'établir des statistiques relatives à la mémoire dans un format conçu pour des systèmes antérieurs, plus simples, aboutit à des données faussées.

svmon

svmon fournit des comptes rendus de l'utilisation de la mémoire au niveau du processus global et au niveau du segment. **–G** et **–P** sont les options les plus intéressantes pour l'optimisation.

–G	Rend compte de l'utilisation de la mémoire pour le système entier.
–P	Rend compte de l'utilisation de la mémoire pour un ou plusieurs processus.

Sous AIX version 4.1, la commande **svmon** est intégrée à la boîte à outils PTX (Performance Toolbox for AIX). Pour savoir si **svmon** est disponible, entrez :

```
lslpp -lI perfagent.tools
```

Si ce module est installé, **svmon** est disponible.

Exemples de sortie vmstat, ps et svmon

L'exemple suivant illustre la sortie de ces commandes sur un grand système. **vmstat** a été exécuté dans une fenêtre séparée, et **ps** et **svmon** successivement. La (première) ligne du relevé **vmstat** a été supprimée :

```
$ vmstat 5
procs    memory             page                  faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 25270  2691   0   0   0   0    0   0 142 2012  41  4 11 86  0
 1  0 25244  2722   0   0   0   0    0   0 138 6752  39 20 70 10  0
 0  0 25244  2722   0   0   0   0    0   0 128   61  34  0  1 99  0
 0  0 25244  2722   0   0   0   0    0   0 137  163  41  1  4 95  0
```

Le compte rendu global **svmon** ci-dessous indique les chiffres correspondants. Le nombre présenté par **vmstat** comme mémoire virtuelle active(avm) est reporté par **svmon** comme emplacements d'espace de pagination en cours d'utilisation (25270). Le nombre de trames de page de la liste des disponibilités (2691) est identique dans les deux comptes rendus. Le nombre de pages fixes (2157) fait l'objet d'un compte-rendu séparé car les pages fixes sont comptées dans les pages en cours d'utilisation.

```
$ svmon -G
       m e m o r y          i n  u s e          p i n        p g  s p a c e
  size inuse  free   pin   work  pers  clnt   work  pers  clnt   size inuse
24576  21885  2691  2157  13172  7899   814   2157     0     0  40960 25270
```

Pour faire ressortir un processus donné, long, sur cette machine, nous pouvons comparer les comptes rendus **ps v** et **svmon –P**. Le programme réel a été renommé **anon**.

```
$ ps v 35851
   PID  TTY STAT  TIME PGIN  SIZE   RSS  LIM  TSIZ   TRS %CPU %MEM COMMAND
 35851    - S     0:03  494  1192  2696   xx  1147  1380  0.2  3.0   anon
```

La valeur de SIZE (1192) est le nombre de Pgspace de **svmon** (298) multiplié par quatre. RSS (2696) est égal à la somme du nombre de pages du segment privé du processus (329) et du nombre de pages du segment du code (345) multipliée par quatre. TSIZE n'est pas lié à l'utilisation de la mémoire réelle. TRS (1380) est égal au nombre de pages utilisées dans le segment du code (345) multiplié par quatre. %MEM est égal à la valeur RSS divisée par la taille de la mémoire réelle en ko, multipliée par 100 et arrondie.

```
$ svmon -P 35851
  Pid                          Command        Inuse        Pin      Pgspace
35851                          anon            2410          2         4624
Pid:  35851
Command:  anon
Segid  Type  Description         Inuse  Pin  Pgspace   Address Range
 18a3  pers  /dev/hd2:5150           1    0        0    0..0
 9873  pers  /dev/hd2:66256          1    0        0    0..0
 4809  work  shared library       1734    0     4326    0..4668 : 60123..65535
 748e  work  private                329    2      298    0..423 : 65402..65535
 2105  pers  code,/dev/hd2:4492    345    0        0    0..402
```

Au fur et à mesure de l'analyse des différents processus de cet environnement, nous observons que la bibliothèque partagée l'est par presque tous les processus du système, et que ses besoins en mémoire font partie de la charge système. La mémoire du segment 9873, également très sollicitée, peut y être incluse. Pour estimer les besoins en mémoire du programme **anon**, la formule serait la suivante :

La mémoire requise pour **anon** est égale à 345 * 4 ko pour le texte programme (partagé par tous les utilisateurs), plus nombre estimé d'utilisateurs simultanés de **anon** multiplié par la somme de la taille du segment de travail (329 * 4 ko) et de 4 ko pour le segment mappé (ID segment 18a3 dans cet exemple).

Programmes à perte de mémoire

Une *perte de mémoire* est un bogue logiciel qui fait qu'un programme affecte de la mémoire à plusieurs reprises, l'utilise puis oublie de la libérer. S'il s'agit d'un programme long (une application interactive, par exemple), le problème peut être sérieux car la mémoire risque d'être fragmentée, et des pages remplies d'informations parasites risquent de s'accumuler en mémoire réelle et en espace de pagination. Il s'est déjà produit que des programmes manquent d'espace de pagination à cause d'une perte de mémoire dans un seul programme.

svmon détecte les pertes de mémoire en recherchant les processus qui contiennent des segments de travail en augmentation continuelle. Il est plus difficile, en particulier pour les applications AIXwindows, d'identifier les sous-routines ou les lignes de codes qui génèrent un grand nombre d'appels **malloc** et **free**. Il existe des programmes de fournisseurs tiers pour analyser les pertes de mémoire, mais ils nécessitent l'accès au code source du programme.

Certaines utilisations de **realloc** peuvent produire le même effet qu'une perte de mémoire. Si un programme exploite fréquemment **realloc** pour augmenter la taille d'une zone de données, le segment de travail d'un processus peut devenir de plus en plus fragmenté si la mémoire libérée par **realloc** ne peut pas être réutilisée. (L'annexe F, "Gestion de la mémoire des applications" contient des informations sur **malloc** et **realloc**.)

Généralement, la mémoire devenue inutile doit être libérée via **free**, *si* la mémoire doit être réutilisée par le programme. D'un autre côté, c'est une perte de temps CPU que de libérer (**free**) de la mémoire après le dernier **malloc**. Lorsque l'exécution du programme est terminée, son segment de travail est détruit et les trames de pages de mémoire réelle qui contiennent les données du segment de travail, ajoutées à la liste des disponibilités.

Voir aussi

Commandes **vmstat**, **ps** et **svmon**.

Analyse de l’exploitation de la mémoire via BigFoot

Remarque : Cette section ne concerne que les versions 4.1 et ultérieures d’AIX.

BigFoot fait partie de Performance Toolbox for AIX. Pour savoir s’il est disponible, entrez :

```
lslpp -lI perfagent.tools
```

Si ce module est installé, BigFoot est disponible.

BigFoot recueille les informations relatives à l’espace occupé en mémoire par un programme en cours. Il établit un compte rendu sur les pages de mémoire virtuelle touchées par le processus. BigFoot est constitué de deux commandes :

bf

collecte des informations sur les pages touchées au cours de l’exécution d’un programme. Il génère des données complètes sur l’exploitation dans le fichier **__bfrpt**.

bfrpt

filtre le fichier **__bfrp**t pour extraire les références mémoire créées par un processus donné.

Estimation de la mémoire requise via rmss

rmss (Reduced-Memory System Simulator) permet de simuler les ESCALA avec des tailles de mémoire inférieures à celles de la machine, sans déposer ni reposer de cartes mémoire. De plus, **rmss** permet d'exploiter une application avec différentes tailles de mémoire, en affichant, pour chaque taille, des statistiques de performance sur, par exemple, le temps de réponse de l'application et la quantité de pagination. En bref, **rmss** est conçu pour répondre à la question : "Combien de méga-octets de mémoire réelle requiert un ESCALA pour exploiter AIX et une application donnée avec un niveau de performance acceptable ?" ou, dans un contexte multi-utilisateur, "Combien d'utilisateurs peuvent exploiter simultanément cette application sur un système disposant de X mégaoctets de mémoire réelle ?"

Sous AIX version 4.1, la commande **rmss** est intégrée à la boîte à outils PTX (Performance Toolbox for AIX). Pour savoir si **rmss** est disponible, entrez :

```
lslpp -lI perfagent.tools
```

Si ce module est installé, **rmss** est disponible.

Il est important de ne pas oublier que la taille mémoire simulée par **rmss** est la taille globale de la mémoire réelle de la machine, dont la mémoire utilisée par AIX et les autres programmes en cours. Il ne s'agit pas de la quantité de mémoire utilisée spécifiquement par l'application elle-même. Compte tenu des dégradations de performances qu'elle peut provoquer, **rmss** est exclusivement réservée à l'utilisateur `root` et aux membres du groupe système. Les sections suivantes décrivent **rmss** plus en détail :

- "Méthodes d'exploitation de **rmss"**
- "Exploitation de **rmss** pour modifier la taille de la mémoire et quitter"
- "Exploitation de **rmss** pour lancer une commande avec des tailles de mémoire différentes"
- "Règles d'exploitation de **rmss**".

Méthodes d'exploitation de rmss

rmss peut être activé : (1) pour modifier la taille de la mémoire réelle et quitter ; (2) comme programme pilote, pour exécuter une application avec différentes tailles de mémoire réelle et afficher des statistiques sur les performances pour chaque taille. La première technique donne des informations sur le comportement d'une application dans un système d'une certaine taille, lorsque l'application est trop complexe pour être exprimée comme une commande unique, ou lorsque vous souhaitez exploiter des instances multiples d'une application. La seconde technique est appropriée lorsque l'application peut être activée comme exécutable ou fichier script shell.

> **Remarque :** Avant de lancer **rmss**, nous vous conseillons de lancer la commande **schedtune –h 0**, page 7-14 pour désactiver le contrôle de charge mémoire VMM. Faute de quoi, ce contrôle risque d'interférer avec vos mesures, sur des mémoires de petite taille. A la fin des essais, restaurez les paramètres de contrôle de charge mémoire à leur valeur antérieure (ou lancez **schedtune –D** pour revenir aux valeurs par défaut).

Exploitation de rmss pour modifier la taille de la mémoire et quitter

Pour modifier la taille de la mémoire et quitter, utilisez l'indicateur **–c** :

```
# rmss -c taille-mémoire
```

Par exemple pour faire passer la taille mémoire à 12 Mo, entrez :

```
# rmss -c 12
```

`taille-mémoire` est un nombre entier ou décimal de méga–octets (12,25, par exemple). En outre, `taille-mémoire` doit être compris entre 4 Mo et la taille physique réelle de la mémoire de votre machine. Selon la configuration matérielle et logicielle, il se peut que **rmss** ne permette pas de réduire la taille mémoire à moins de 8 Mo, à cause de la taille de

la structure des systèmes inhérents tels que le noyau. Si une taille de mémoire donnée est refusée par **rmss**, un message d'erreur est affiché.

rmss diminue la taille effective de la mémoire d'un ESCALA en volant des trames de page disponibles dans la liste tenue par VMM. Ces trames, conservées dans un fond de trames inutilisables, sont renvoyées dans la liste des disponibilités lorsque la taille de la mémoire effective doit être augmentée. De plus, **rmss** ajuste dynamiquement certaines structures de données et variables système qui doivent rester proportionnelles à la taille de la mémoire effective.

La modification de la taille de la mémoire peut prendre jusqu'à 15 ou 20 secondes. En général, plus la réduction souhaitée est importante, plus **rmss** prend de temps. Lorsque **rmss** a abouti, le message suivant s'affiche :

```
Simulated memory size changed to 12,00 Mb.
```

Pour afficher la taille actuelle de la mémoire, utilisez l'indicateur **–p** :

```
# rmss –p
```

rmss répond par :

```
Simulated memory size is  12.00 Mb.
```

Si, finalement, vous souhaitez régler la taille de la mémoire sur celle de la mémoire actuelle de la machine, utilisez l'indicateur **–r** :

```
# rmss –r
```

Quelle que soit la taille de la mémoire virtuelle actuelle, l'indicateur **–r** ramène la taille de la mémoire à celle de la mémoire réelle physique de la machine. Dans cet exemple, où un système de 16 Mo est exploité, **rmss** répond :

```
Simulated memory size changed to 16,00 Mb.
```

Remarque : La commande **rmss** indique la quantité de mémoire réelle *utilisable*. Sur les machines dont la mémoire n'est pas fiable ou déjà utilisée, **rmss** indique la quantité de mémoire réelle, c'est-à-dire la mémoire physique réelle moins la mémoire non fiable ou déjà utilisée par le système. Par exemple, la commande **rmss –r** peut indiquer :

```
Simulated memory size changed to 79.9062 Mb.
```

Dans cet exemple, une partie de la mémoire est considérée comme non fiable ou est réservée par un périphérique (et n'est donc pas disponible pour l'utilisateur).

Indicateurs –c, –p et –r

Les indicateurs **–c**, **–p** et **–r** de **rmss** permettent, contrairement à d'autres options, d'exploiter des applications complexes qui ne peuvent être exprimées en un fichier exécutable ou un script shell unique. Les indicateurs **–c**, **–p** et **–r** vous obligent, par contre, à évaluer vous-même les performances. Il existe un moyen simple de le faire : **vmstat –s** mesure l'activité au niveau de l'espace de pagination lors de l'exécution d'une application.

Si vous lancez **vmstat –s**, puis l'application, puis à nouveau **vmstat –s**, et calculez la différence entre le nombre de transferts en mémoire d'espace de pagination "avant" et "après", vous obtenez le nombre de transferts effectués pendant le programme. En outre, en chronométrant le programme, puis en divisant le nombre de transferts par la durée d'exécution du programme, vous obtenez le taux moyen de transferts de l'espace de pagination.

Il est également important d'exécuter plusieurs fois l'application avec chaque taille mémoire. Deux raisons : d'abord, lors de la modification de la taille de la mémoire, **rmss** vide une grande quantité de mémoire. Ainsi, la première fois que l'application est lancée après modification de la taille de la mémoire, il est possible qu'une partie importante du temps d'exécution soit pris par la lecture des fichiers en mémoire réelle. Cependant, les fichiers pouvant rester en mémoire après l'exécution de l'application, les temps d'exécution suivants peuvent être plus courts. L'autre raison d'exécuter plusieurs fois l'application est d'avoir une idée de ses performances *moyennes* avec une taille donnée de mémoire. Le ESCALA et

AIX sont des systèmes complexes, et il est impossible de dupliquer l'état du système chaque fois que votre application est exécutée. Ainsi, la performance d'une application peut varier de façon significative d'une exécution à l'autre.

En résumé, vous devez tenir compte des étapes suivantes pour activer **rmss** :

```
s'il existe plusieurs tailles de mémoire à examiner :
  {
  changez la taille de la mémoire via rmss -c ;

  lancez l'application une première fois ;
  pour plusieurs exécutions :
    {
    lancez vmstat -s pour obtenir le nombre "avant" de transferts
    en mémoire d'espace de pagination ;
    lancez et chronométrez l'application ;
    lancez vmstat -s pour obtenir le nombre "après" de transferts
    en mémoire d'espace de pagination ;
    soustrayez la valeur "avant" de la valeur "après" pour
       obtenir le nombre de transferts en mémoire pendant
    l'exécution ;
    divisez le nombre de transferts en mémoire d'espace de pagination par
le temps de réponse
       pour obtenir le taux de transferts en mémoire de l'espace
    de pagination ;
    }
  }

lancez rmss -r pour restaurer la taille mémoire normale (ou réamorcez)
```

Le calcul du nombre d'E/S de pagination (après – avant) peut être automatisé via le script **vmstat.sh** inclus dans le module PerfPMR.

Exploitation de rmss pour lancer une commande avec des tailles de mémoire différentes

Les indicateurs **–s**, **–f**, **–d**, **–n** et **–o** sont exploités pour activer **rmss** comme programme pilote : **rmss** exécute une application spécifiée avec des tailles de mémoire différentes et affiche des statistiques sur ses performances pour chaque taille de mémoire. La syntaxe pour ce type d'activation de **rmss** est la suivante :

```
rmss [ -s smemsize ] [ -f fmemsize ] [ -d memdelta ]
     [ -n numiterations ] [ -o outputfile ] command
```

L'indicateur **–n** permet de spécifier le nombre d'exécutions et mesure le temps d'exécution de la commande pour chaque taille de mémoire. **–o** spécifie le fichier dans lequel écrire le compte rendu **rmss** ; *command* est l'application à lancer et à chronométrer pour chaque taille de mémoire. Ces indicateurs sont détaillés plus loin.

Les indicateurs **–s**, **–f** et **–d** permettent de spécifier les différentes tailles de mémoire : **–s** la taille de départ, **–f** la taille finale et **–d** la différence entre les tailles. Toutes ces valeurs sont des nombres entiers ou décimaux, en méga–octets. Par exemple, pour lancer et mesurer une commande avec 24, 20, 16, 12 et 8 Mo, utilisez la combinaison suivante :

```
-s 24 -f 8 -d 4
```

De même, pour les tailles 16, 24, 32, 40 et 48 Mo, la combinaison est la suivante :

```
-s 16 -f 48 -d 8
```

Si vous omettez l'indicateur **–s**, **rmss** démarre avec la taille mémoire actuelle de la machine. Si vous omettez l'indicateur **–f**, **rmss** s'arrête à 8 Mo. Si vous omettez l'indicateur **–d**, la valeur par défaut (8 Mo) est appliquée.

Quelles sont les valeurs possibles pour **–s**, **–f** et **–d** ? Le plus simple consiste à choisir les tailles mémoire des ESCALA censés faire tourner l'application concernée. Cependant, une incrémentation inférieure à 8 Mo peut être utile, car elle vous donne une idée de la marge de manœuvre dont vous disposez lorsque vous fixez votre choix sur une taille.
Par exemple, si une application donnée s'emballe à 8 Mo mais fonctionne sans transferts

de pagination à 16 Mo, il est intéressant de savoir à quelle taille, entre 8 et 16 Mo, l'application commence à s'emballer. Si elle démarre à 15 Mo, vous pouvez configurer le système avec plus de 16 Mo de mémoire, ou modifier l'application de sorte que la marge de manœuvre soit plus importante. Mais, si elle démarre à 9 Mo, vous savez que vous disposez d'une marge suffisante avec une machine à 16 Mo.

L'indicateur **–n** spécifie le nombre d'exécutions et de mesures d'une commande pour chaque taille de mémoire. Après avoir lancé et mesuré l'application autant de fois que spécifié, **rmss** affiche des statistiques sur les performances moyennes de l'application pour une taille donnée. Pour lancer la commande 3 fois pour chaque taille de mémoire, entrez :

```
-n 3
```

Si **–n** est omis, **rmss** détermine au cours de l'initialisation le nombre de fois où l'application doit être exécutée pour accumuler un temps d'exécution total de 10 secondes. **rmss** effectue cette opération pour assurer que les statistiques relatives aux performances des programmes courts ne seront pas faussées par des éléments externes (démons, par exemple).

> **Remarque :** Si vous mesurez un programme très bref, le nombre d'exécutions nécessaires pour accumuler 10 secondes de temps CPU peut être très important. Chaque exécution du programme mobilisant environ, et au moins, 2 secondes de charge **rmss**, il est préférable de spécifier explicitement le paramètre **–n** pour les programmes courts.

Quelles sont les valeurs recommandées pour **–n** ? Si vous savez que l'application s'exécute en plus de 10 secondes, vous pouvez spécifier `-n 1` pour que la commande ne soit exécutée et mesurée qu'une seule fois pour chaque taille mémoire. Il est intéressant d'utiliser l'indicateur **–n** car, au cours de l'initialisation, **rmss** n'a pas à déterminer le nombre d'exécutions du programme. Ceci est encore plus intéressant lorsque les commandes à mesurer sont longues et interactives.

Il ne faut pas oublier que **rmss** exécute toujours la commande une première fois pour chaque taille mémoire, à titre de mise en route, avant de la lancer et de la mesurer. Ceci pour éviter les E/S qui se produisent normalement lorsque l'application ne se trouve pas déjà en mémoire. Bien que de telles E/S affectent les performances, cela n'est pas forcément dû à un manque de mémoire réelle. La mise en route n'est pas comptée dans le nombre spécifié par l'indicateur **–n**.

L'indicateur **–o** permet de spécifier un fichier dans lequel écrire le rapport **rmss**. Si l'indicateur **–o** est omis, le rapport est écrit dans le fichier `rmss.out`.

command spécifie l'application à mesurer : *command* est un exécutable ou un script shell, avec ou sans arguments sur la ligne de commande. Il existe toutefois quelques contraintes sur la forme de cette commande. Elle ne doit pas contenir de réacheminement de l'entrée ou de la sortie (`foo > output`, `foo < input`, par exemple). Ceci parce que **rmss** considère tout ce qui se trouve à droite du nom de la commande comme des arguments de cette commande. Pour effectuer un réacheminement, vous devez placer la commande dans un fichier script shell.

Normalement, pour enregistrer la sortie de **rmss** dans un fichier spécifique, vous utilisez l'option **–o**. Pour réacheminer, par exemple, la sortie `stdout` de **rmss** (pour l'insérer, par exemple, à la fin d'un fichier existant), vous devez, avec le shell Korn, mettre l'appel de **rmss** entre parenthèses, comme suit :

```
# (rmss -s 24 -f 8 foo) >> sortie
```

Interprétation des résultats de rmss

Cette section propose quelques interprétations des statistiques de performance produites par **rmss**. Commençons par quelques résultats typiques.

L'exemple "Compte rendu généré pour le programme foo", page 7-10 a été généré en lançant **rmss** sur un programme réel, dont le nom été simplement modifié en `foo` pour

préserver l'anonymat. La commande normale pour générer ce compte-rendu est la suivante :

```
# rmss -s 16 -f 8 -d 1 -n 1 -o rmss.out foo
```

Compte rendu généré pour le programme foo

```
Hostname:  widgeon.austin.ibm.com
Real memory size:   16,00 Mb
Time of day:  Thu Jan  8 19:04:04 1990
Command:  foo

Simulated memory size initialized to  16,00 Mb.

Number of iterations per memory size = 1 warm-up + 1 measured = 2.

Memory size   Avg. Pageins  Avg. Response Time  Avg. Pagein Rate
(megabytes)                      (sec.)          (pageins / sec.)
-----------------------------------------------------------------
16.00            115.0           123.9                0.9
15.00            112.0           125.1                0.9
14.00            179.0           126.2                1.4
13.00             81.0           125.7                0.6
12.00            403.0           132.0                3.1
11.00            855.0           141.5                6.0
10.00           1161.0           146.8                7.9
9.00            1529.0           161.3                9.5
8.00            2931.0           202.5               14.5
```

Ce compte rendu comporte quatre colonnes : la première indique la taille de la mémoire, et la colonne `Avg. Pageins` le nombre moyen de transferts en mémoire au cours de l'exécution de l'application avec cette taille de mémoire. Il est important de noter que `Avg. Pageins` fait référence à toutes les opérations de transfert en mémoire, y compris le code, les données et les fichiers lus à partir de tous les programmes, qui se sont déroulées en même temps que l'application. La colonne `Avg. Response Time` indique la durée moyenne d'exécution de l'application, tandis que `Avg. Pagein Rate` indique le taux moyen de transferts de pages.

Etudions d'abord la colonne `Avg. Pagein Rate`. Entre 16 et 13 Mo, le taux de transfert est assez réduit (< 1,5 transfert/s). Entre 13 et 8 Mo, ce taux augmente, d'abord graduellement, puis plus rapidement lorsque l'on s'approche de 8 Mo. La colonne `Avg. Response Time` a la même structure : relativement bas au départ, puis de plus en plus important au fur et à mesure que la taille de la mémoire tend vers 8 Mo.

Ici, le taux de transfert en mémoire décroît avec le passage de la taille de la mémoire de 14 Mo (1,4 transfert/s) à 13 Mo (0,6 transfert/s). Il n'y a pas de quoi s'alarmer : il est impossible d'obtenir des résultats uniformes avec un système réel. L'essentiel est que le taux de transfert en mémoire reste relativement bas pour une mémoire de 14 Mo et de 13 Mo.

Nous pouvons déduire quelques remarques de ce compte rendu. Si la performance d'une application est jugée inacceptable à 8 Mo (ce qui est probable), le fait d'ajouter de la mémoire améliore cette performance de façon significative. Le temps de réponse varie entre 124 secondes pour 16 Mo et 202 secondes pour 8 Mo, ce qui constitue une augmentation de 63 %. D'un autre côté, si la performance est jugée inacceptable à 16 Mo, l'ajout de mémoire ne l'améliore pas beaucoup car, à cette taille, les transferts en mémoire ne ralentissent pas le programme de façon significative.

Exemples d'exploitation des indicateurs –s, –f, –d, –n et –o

Pour connaître les performances du script shell `ccfoo` qui comprend la commande `cc -O -c foo.c` avec 16, 14, 12, 10, 8 et 6 Mo de mémoire, lancer et mesurer la commande deux fois pour chaque taille de mémoire, puis écrire le compte rendu dans le fichier `cc.rmss.out`, entrez :

```
# rmss -s 16 -f 6 -d 2 -n 2 -o cc.rmss.out ccfoo
```

Compte rendu pour cc

La sortie de ce compte rendu est la suivante :

```
Hostname:  terran
Real memory size:   32,00 Mb
Time of day:  Mon Apr 20 16:23:03 1992
Command:  ccfoo

Simulated memory size initialized to  16,00 Mb.

Number of iterations per memory size = 1 warm-up + 2 measured = 3.

Memory size    Avg. Pageins   Avg. Response Time   Avg. Pagein Rate
(megabytes)                         (sec.)         (pageins / sec.)
-------------------------------------------------------------------
16.00              0.0             0.4                   0.0
14.00              0.0             0.4                   0.0
12.00              0.0             0.4                   0.0
10.00              0.0             0.4                   0.0
8.00               0.5             0.4                   1.2
6.00               786.0          13.5                  58.4

Simulated final memory size.
```

Ce compte rendu montre que la modification est insuffisante. Les performances se dégradent avec une machine de 6 Mo, mais restent sensiblement les mêmes pour les tailles supérieures. Nous pouvons réaliser une nouvelle mesure avec une fourchette de tailles de mémoire moins importante et un delta inférieur comme suit :

```
rmss -s 11 -f 5 -d 1 -n 2 ccfoo
```

Cela nous donne une image plus précise de la courbe du temps de réponse du compilateur pour ce programme :

```
Hostname:  terran
Real memory size:   32,00 Mb
Time of day:  Mon Apr 20 16:11:38 1992
Command:  ccfoo

Simulated memory size initialized to  11,00 Mb.

Number of iterations per memory size = 1 warm-up + 2 measured = 3.

Memory size    Avg. Pageins   Avg. Response Time   Avg. Pagein Rate
(megabytes)                         (sec.)         (pageins / sec.)
-------------------------------------------------------------------
11.00              0.0             0.4                   0.0
10.00              0.0             0.4                   0.0
9.00               0.5             0.4                   1.1
8.00               0.0             0.4                   0.0
7.00               207.0           3.7                  56.1
6.00               898.0          16.1                  55.9
5.00               1038.0         19.5                  53.1

Simulated final memory size.
```

Compte rendu pour une copie distante de 16 Mo

L'exemple suivant présente un compte rendu généré (sur une machine client) en lançant **rmss** sur une commande qui copie un fichier de 16 Mo depuis une machine distante (un serveur) via NFS.

```
Hostname:  xray.austin.ibm.com
Real memory size:   48.00 Mb
Time of day:  Mon Aug 13 18:16:42 1990
Command:  cp /mnt/a16Mfile /dev/null

Simulated memory size initialized to  48.00 Mb.

Number of iterations per memory size = 1 warm-up + 4 measured = 5.

Memory size    Avg. Pageins   Avg. Response Time   Avg. Pagein Rate
(megabytes)                     (sec.)           (pageins / sec.)
-----------------------------------------------------------------
48.00              0.0            2.7                   0.0
40.00              0.0            2.7                   0.0
32.00              0.0            2.7                   0.0
24.00              1520.8        26.9                  56.6
16.00              4104.2        67.5                  60.8
8.00               4106.8        66.9                  61.4
```

Remarquez que le temps de réponse et le taux de transfert en mémoire de ce compte rendu démarrent avec une valeur assez basse, augmentent rapidement jusqu'à 24 Mo de mémoire avant d'atteindre un palier à 16 et 8 Mo. Ce compte rendu illustre l'importance du choix d'une fourchette de mémoire assez large lors de l'exploitation de **rmss**. Si l'utilisateur n'avait considéré que les tailles de mémoire entre 24 et 8 Mo, il n'aurait pas eu l'opportunité d'affecter suffisamment de mémoire pour que l'application fonctionne sans transfert en mémoire.

Compte rendu de find / –ls >/dev/null

L'exemple suivant est un compte rendu généré en lançant **rmss** sur le script shell `findbench.sh`, comportant la commande `find / -ls > /dev/null`, qui effectue un **ls** de chaque fichier du système. La commande à l'origine du compte rendu est la suivante :

```
# rmss -s 48 -d 8 -f 4.5 -n 1 -o find.out findbench.sh
```

La taille de la mémoire finale (4,5 Mo) a été choisie car c'est la plus petite taille mémoire accessible via **rmss** sur cette machine.

```
Hostname:  xray.austin.ibm.com
Real memory size:   48.00 Mb
Time of day:  Mon Aug 13 14:38:23 1990
Command:  findbench.sh

Simulated memory size initialized to  48.00 Mb.

Number of iterations per memory size = 1 warm-up + 1 measured = 2.

Memory size    Avg. Pageins   Avg. Response Time   Avg. Pagein Rate
(megabytes)                     (sec.)           (pageins / sec.)
-----------------------------------------------------------------
48.00              373.0          25.5                  14.6
40.00              377.0          27.3                  13.8
32.00              376.0          27.5                  13.7
24.00              370.0          27.6                  13.4
16.00              376.0          27.3                  13.8
8.00               370.0          27.1                  13.6
4.50               1329.0         57.6                  23.1
```

Comme dans le premier exemple, les temps de réponse moyens et les valeurs du taux de transfert en mémoire restent presque stables avec la diminution de la taille de la mémoire jusqu'à 4,5 Mo, seuil à partir duquel les deux paramètres augmentent énormément. Cependant, le taux de transfert en mémoire est relativement élevé (presque 14 transferts/s) à partir de 48 Mo et jusqu'à 8 Mo. La conclusion est que, avec certaines applications, aucune taille de mémoire n'est dans la pratique suffisante pour éliminer les transferts en

mémoire, car les programmes sont eux–mêmes très consommateurs d'E/S. Parmi les programmes courants très consommateurs d'E/S, se trouvent les programmes qui balayent ou accèdent en mode aléatoire à de nombreuses pages de fichier très volumineux.

Conseils d'utilisation des indicateurs –s, –f, –d, –n et –o

L'une des fonctions utiles de **rmss**, exploité de cette façon, est qu'il peut être interrompu (par la touche d'interruption, `Ctrl-C`, par défaut) sans détruire le compte rendu écrit dans le fichier de sortie. Outre l'écriture du compte rendu dans le fichier sortie, **rmss** redéfinit la taille de la mémoire à celle de la mémoire physique de la machine.

Vous pouvez lancer **rmss** en arrière-plan, même après vous être déconnecté, via la commande **nohup**. Pour ce faire, faites précéder la commande **rmss** de `nohup`, et terminez la commande par un perluète (&) :

```
# nohup rmss –s 48 –f 8 –o foo.out foo &
```

Règles d'exploitation de rmss

Quelle que soit la méthode utilisée pour activer **rmss**, il est important de recréer l'environnement de l'utilisateur final aussi précisément que possible. Par exemple, utilisez-vous le même modèle de CPU ? Le même modèle de disques ? Le même réseau ? Les utilisateurs disposeront-ils de fichiers d'application montés depuis un nœud distant via NFS ou d'un autre système de fichiers distribué ? Ce dernier point est particulièrement important car les pages des fichiers distants ne sont pas traitées de la même façon par VMM que celles des fichiers locaux.

De même, il est préférable d'éliminer toute activité système qui n'est pas liée à la configuration souhaitée ou à l'application que vous souhaitez mesurer. Il convient, par exemple, d'interdire à d'autres personnes de travailler sur la machine où **rmss** est actif, sauf si elles traitent une partie de la charge de travail à mesurer.

Remaque : Il est impossible de lancer plusieurs appels simultanés de **rmss**.

Une fois toutes les exécutions **rmss** terminées, mieux vaut mettre le système hors tension et le réamorcer. Cette opération supprime toutes les modifications effectuées par **rmss** sur le système et restaure les paramètres habituels de contrôle de charge mémoire.

Optimisation du contrôle de charge mémoire VMM

L'utilitaire de contrôle de charge de mémoire VMM, décrit page 2-8, permet d'éviter l'emballement des systèmes surchargés, qui génère une paralysie continue et dans laquelle les processus du système passent leur temps à se voler des trames de mémoire et à lire ou écrire des pages sur l'unité de pagination.

Optimisation du contrôle de charge mémoire

Le contrôle de charge mémoire est conçu pour aplanir les pointes de charge *peu fréquentes* qui aboutissent à l'emballement du système. Il *n'a pas* pour objet de pallier en permanence les défaillances d'un système insuffisamment doté en RAM. Il s'agit de la sécurité du réseau, ce n'est pas un trampoline. La seule solution à une insuffisance chronique, persistante de RAM est d'ajouter de la RAM, et non de "bricoler" – via le contrôle de charge mémoire – des temps de réponse acceptables. Par contre, le contrôle de charge mémoire doit vraiment être optimisé dans les cas où la quantité de RAM est *supérieure* aux valeurs par défaut, c'est-à-dire dans les configurations où les valeurs par défaut sont trop modérées.

Ne modifiez pas les paramètres du contrôle de charge mémoire sauf si la charge de travail est importante et que les paramètres par défaut risquent de ne plus suffire.

Les paramètres par défaut d'origine du système sont toujours en vigueur sauf s'ils ont été modifiés, alors que les paramètres modifiés ne perdurent pas après un réamorçage. Toutes les activités d'optimisation du contrôle de charge mémoire doivent être effectuées par un utilisateur `root`. L'administrateur système peut modifier des paramètres pour "adapter" l'algorithme à une charge de travail particulière ou les désactiver entièrement. C'est l'objet de la commande **schedtune**. Le code source et objet de **schedtune** résident dans **/usr/samples/kernel**.

> **Attention : schedtune** réside dans le répertoire **samples** car il dépend *étroitement* de l'implantation de VMM. Le code **schedtune** est propre à chaque version d'AIX : il a été mis au point pour le VMM de cette version. L'exploitation d'une version de **schedtune** sur une autre version peut aboutir à un incident du système d'exploitation. En outre, les fonctions de **schedtune** peuvent varier d'une version à l'autre. Il est déconseillé d'utiliser des scripts shell ou des entrées **inittab** comprenant **schedtune** sur une nouvelle version sans consulter la documentation **schedtune** de cette nouvelle version. **schedtune** n'est pas pris en charge sous SMIT, et toutes les combinaisons de paramètres n'ont pas été testées.

schedtune –? affiche une brève description des indicateurs et des options. **schedtune** sans indicateur affiche les paramètres courants, comme suit :

```
      THRASH            SUSP        FORK       SCHED
-h     -p     -m       -w     -e       -f         -t
SYS   PROC   MULTI    WAIT   GRACE    TICKS   TIME_SLICE
 6    4      2         1       2       10     0
```

(Les indicateurs **–f** et **–t** ne font pas partie du mécanisme de contrôle de charge mémoire. Les renseignements les concernant se trouvent avec la description de la syntaxe de **schedtune**. Il est également question de l'indicateur **–t** dans "Modification de la tranche horaire du programmateur", page 6-25.) Après un essai d'optimisation, le contrôle de charge mémoire peut être restauré à ses valeurs par défaut via **schedtune –D**.

Pour désactiver le contrôle de charge mémoire, affectez une valeur au paramètre telle que les processus ne soient jamais interrompus. Ainsi, **schedtune –h 0** affecte au seuil d'emballement une valeur excessive : l'algorithme n'en détecte plus.

Dans certains cas, il est préférable de désactiver le contrôle de charge mémoire dès le début. Par exemple, si vous exploitez un émulateur de terminal avec fonction de dépassement de délai pour simuler une charge de travail multi-utilisateur, l'intervention d'un contrôle de charge mémoire peut retarder suffisamment les réponses pour que le processus

soit tué par cette fonction. Si vous lancez **rmss** pour connaître les effets de la diminution des tailles de mémoire, il est préférable de désactiver le contrôle de charge mémoire pour éviter qu'il n'interfère avec vos mesures.

Si la désactivation du contrôle de charge mémoire augmente le nombre de situations d'emballement (avec un temps de réaction amoindri), alors le contrôle de charge mémoire joue un rôle actif de soutien dans le système. Dans ce cas, l'optimisation des paramètres de contrôle de charge mémoire – ou l'ajout de mémoire RAM – peut améliorer les performances.

La définition du niveau de multiprogrammation minimum, *m*, évite aux *m* processus d'être suspendus. Supposons qu'un administrateur système sache qu'au moins dix processus doivent toujours être résidants et actifs en RAM pour de bonnes performances, et soupçonne le contrôle de charge mémoire de suspendre les processus de façon trop brutale. Si **schedtune –m 10** est lancé, le système ne suspendra jamais tant de processus que moins de dix processus restent en compétition pour accéder à la mémoire.
Le paramètre *m* ne tient pas compte du noyau, des processus fixés en RAM par l'appel système **plock**, des processus à priorité fixée à une valeur inférieure à 60 et à ceux en attente d'événements. La valeur par défaut du système *m*=2 permet d'assurer que le noyau, les processus fixés et deux processus utilisateurs feront toujours partie des processus en compétition pour l'accès à la RAM.

La valeur *m=2*, adaptée aux configurations mono-utilisateur de bureau, est souvent insuffisante pour les configurations multiutilisateurs plus importantes ou les serveurs disposant de grandes quantités de RAM. Sur ces systèmes, affecter la valeur 4 ou 6 à *m* peut améliorer les performances.

Lorsque vous avez déterminé le nombre de processus susceptibles de fonctionner sur le système pendant les pics d'activité, ajoutez **schedtune** à la fin du fichier **/etc/inittab**, pour vous assurer qu'il sera bien exécuté à chaque amorçage du système et que les paramètres par défaut seront restaurés. Par exemple, la ligne **/etc/inittab** suivante fait passer le niveau minimum de multiprogrammation à 4 sous AIX version 4.1 :

```
schedtune:2:wait:/usr/samples/kernel/schedtune -m 4
```

La ligne **/etc/inittab** correspondante sous la version 3.2.5 serait :

```
schedtune:2:wait:/usr/lpp/bos/samples/schedtune -m 4
```

N'oubliez pas que cette ligne ne doit pas être utilisée sur une autre version d'AIX sans consultation préalable de la documentation.

Alors qu'il est possible de modifier les paramètres qui contrôlent le nombre d'interruptions des processus et le critère de sélection des processus à interrompre, il est impossible de prévoir avec précision l'effet de telles modifications sur une configuration et une charge de travail particulières. Le choix des paramètres par défaut est délicat et requiert des outils de mesure sophistiqués et une observation attentive des charges de travail récurrentes. Soyez très prudent si vous envisagez d'ajuster des paramètres de contrôle de charge mémoire autres que ceux détaillés ici.

Optimisation du remplacement de page VMM

L'algorithme de gestion de la mémoire, détaillé page 2-5, tente de maintenir dans les limites spécifiées la taille de la liste des disponibilités et le pourcentage de mémoire réelle occupée par des pages de segment persistant. Ces limites peuvent être modifiées via la commande **vmtune**, qui ne peut être lancée que par un utilisateur `root`.

> **Attention : vmtune** réside dans le répertoire **samples** car il dépend *étroitement* de l'implantation de VMM. Le code **vmtune** qui accompagne chaque version d'AIX a été mis au point spécialement pour cette version de VMM. L'exploitation d'une version de **vmtune** sur une autre version peut aboutir à un incident du système d'exploitation. En outre, les fonctions de **vmtune** peuvent varier d'une version à l'autre. Il est déconseillé de diffuser des scripts shell ou des entrées **inittab** comprenant **vmtune** dans une nouvelle version sans vérifier dans la documentation **vmtune** de la nouvelle version que les scripts conservent l'effet désiré.

Paramètres minfree et maxfree

Le but de la liste des disponibilités est de garder trace des trames de page de la mémoire réelle libérées par la fin de l'exécution des processus et de les fournir immédiatement aux demandeurs, sans les obliger à attendre le vol des pages ni la fin des E/S associées. La limite **minfree** précise la taille de la liste des disponibilités en-deçà de laquelle doit commencer le vol de pages pour réapprovisionner la liste. **maxfree** est la taille au-delà de laquelle le vol prend fin.

L'optimisation de ces limites a les buts suivants :

- assurer à toute activité ayant des objectifs de temps de réponse critiques l'obtention des trames de page nécessaires dans la liste des disponibilités ;
- éviter au système une surcharge d'E/S suite à des vols de pages prématurés pour étendre la liste des disponibilités.

Si vous disposez d'une courte liste des programmes à exécuter rapidement, vous pouvez déterminer leur besoin en mémoire via **svmon** (voir "Quantité de mémoire utilisée", page 7-2), et affecter la plus grande taille à **minfree**. Cependant, cette technique risque d'être trop modérée car toutes les pages utilisées par un processus ne sont pas acquises en une seule rafale. En outre, vous pouvez passer à côté de demandes dynamiques issues de programmes ne figurant pas dans la liste, mais qui peuvent diminuer la taille moyenne de la liste des disponibilités lors de leur exécution.

vmstat est un outil moins précis mais plus complet pour déterminer la valeur appropriée de **minfree**. Vous trouverez ci-après un exemple de sortie **vmstat 1** obtenu lors d'une compilation XLC sur un système autrement inactif. La première ligne a été conservée ; notez qu'elle contient un résumé des mesures CPU ainsi que celles d'autres activités, mais pas de statistiques sur la mémoire courante.

```
procs    memory            page                  faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b    avm   fre  re  pi  po  fr   sr  cy  in    sy  cs us sy id wa
 0  0   3085   118   0   0   0   0    0   0 115     2  19  0  0 99  0
 0  0   3086   117   0   0   0   0    0   0 119   134  24  1  3 96  0
 2  0   3141    55   2   0   6  24   98   0 175   223  60  3  9 54 34
 0  1   3254    57   0   0   6 176  814   0 205   219 110 22 14  0 64
 0  1   3342    59   0   0  42 104  249   0 163   314  57 43 16  0 42
 1  0   3411    78   0   0  49 104  169   0 176   306  51 30 15  0 55
 1  0   3528   160   1   0  10 216  487   0 143   387  54 50 22  0 27
 1  0   3627    94   0   0   0  72  160   0 148   292  79 57  9  0 34
 1  0   3444   327   0   0   0  64  102   0 132   150  41 82  8  0 11
 1  0   3505   251   0   0   0   0    0   0 128   189  50 79 11  0 11
 1  0   3550   206   0   0   0   0    0   0 124   150  22 94  6  0  0
 1  0   3576   180   0   0   0   0    0   0 121   145  30 96  4  0  0
 0  1   3654   100   0   0   0   0    0   0 124   145  28 91  8  0  1
 1  0   3586   208   0   0   0  40   68   0 123   139  24 91  9  0  0
```

Le compilateur n'ayant pas été exécuté récemment, le code du compilateur lui–même doit être chargé. Le compilateur acquiert environ 2 Mo en 6 secondes environ. Sur ce système 32 Mo, la valeur de **maxfree** est de 64 et celle de **minfree** de 56. Le compilateur amène presque instantanément la taille de la liste des disponibilités en dessous de la valeur **minfree**, et pendant quelques secondes une intense activité de vol de page est déployée. Certains de ces vols requièrent l'écriture des pages de segment de travail modifiées dans l'espace de pagination, comme indiqué dans la colonne `po`. Si ces vols occasionnent l'écriture de pages de segments permanents modifiés, cette E/S n'apparaît pas dans le compte rendu **vmstat** (sauf si **vmstat** doit reporter les E/S du/des volume(s) physique(s) dans lesquels sont écrites les pages permanentes).

Le but de cet exemple n'est pas de vous inciter à définir **minfree** à 500 pour prendre en charge des compilations importantes. Il indique seulement comment exploiter **vmstat** pour identifier les situations dans lesquelles la liste des disponibilités doit être réapprovisionnée lorsqu'un programme est en attente d'espace. Dans ce cas, la durée d'exécution du compilateur a été allongée d'environ 2 secondes, car le nombre de trames de page immédiatement disponibles était insuffisant. Si vous analysez la consommation en trames de pages de votre programme, au cours de l'initialisation ou du traitement normal, vous aurez rapidement une idée du nombre de trames de page requises pour que le programme n'attende pas de mémoire.

Lors du choix de la taille de la liste des disponibilités pour la charge de travail interactive, vous pouvez définir **minfree** via **vmtune**. **maxfree** doit être supérieur à **minfree** d'au moins 8 (ou de la valeur de **maxpgahead**, si elle est supérieure). Si nous concluons de l'exemple précédent que **minfree** doit être défini à 128, et que **maxpgahead** est défini à 16 pour améliorer les performances de l'accès séquentiel, nous devons lancer la commande **vmtune** suivante – qui génère la sortie qui suit :

```
# /usr/lpp/bos/samples/vmtune -f 128 -F 144

 minperm  maxperm  minpgahead  maxpgahead  minfree  maxfree  numperm
   1392     5734        2          16          56       64     3106
number of memory frames = 8192   number of bad memory pages = 0
maxperm=70.0% of real memory
minperm=17.0% of real memory

 minperm  maxperm  minpgahead  maxpgahead  minfree  maxfree  numperm
   1392     5734        2          16         128      144     3106
number of memory frames = 8192   number of bad memory pages = 0
maxperm=70.0% of real memory
minperm=17.0% of real memory
```

Paramètres minperm et maxperm

AIX profite de la variation des besoins en mémoire réelle pour laisser en mémoire des pages de fichiers déjà lues ou écrites. Ainsi, si des pages de fichier sont réclamées à nouveau avant que leurs trames de page aient été réaffectées, une opération d'E/S est évitée. (Même si une trame d'une page de fichier a été volée et placée dans la liste des disponibilités, si cette page est réclamée avant que la trame ne soit effectivement utilisée à autre chose, elle sera récupérée dans la liste des disponibilités.) Ces pages de fichier peuvent appartenir à des systèmes de fichiers locaux ou distants (NFS, par exemple).

Le rapport trames utilisées pour les fichiers/trames utilisées pour les segments de calcul (textes de travail ou programme) est plus ou moins contrôlé par **minperm** et **maxperm**.

Pour une charge de travail particulière, il peut être intéressant de privilégier l'absence d'E/S de fichiers. Pour une autre charge, la conservation de segments de calcul en mémoire peut primer. Pour analyser ce rapport dans un environnement non optimisé, lancez la commande **vmtune** sans argument :

```
# vmtune

 minperm  maxperm  minpgahead  maxpgahead  minfree  maxfree  numperm
   1433     5734        2          16         128      144     3497
number of memory frames = 8192   number of bad memory pages = 0
maxperm=70.0% of real memory
minperm=17.5% of real memory
```

Les valeurs par défaut sont calculées par l'algorithme suivant :

```
minperm (en pages) =  ((nombre de trames de mémoire) - 1024) *
0,2
maxperm (en pages) =  ((nombre de trames de mémoire) - 1024) *
0,8
```

La colonne **numperm** donne le nombre de pages de fichier en mémoire, 3497, soit 42,7 % de mémoire réelle. Si la charge de travail exploite peu de fichiers récemment lus ou écrits, il est possible de réduire la quantité de mémoire utilisée à cet effet. La commande :

```
# vmtune -p 15 -P 40
```

définit **minperm** à 15 % et **maxperm** à 40 % de mémoire réelle. Cela permet d'assurer que VMM ne vole les trames des pages de fichier que lorsque le rapport pages de fichier/mémoire totale dépasse 40 %. A l'inverse, si l'application renvoie fréquemment à un jeu réduit de fichiers existants (spécialement si ces fichiers font partie d'un système de fichiers monté NFS), il est possible d'affecter plus d'espace pour la mise en mémoire cache locale des pages de fichier avec :

```
# vmtune -p 30 -P 60
```

Voir aussi

Commandes **schedtune** et **vmtune**.

Chapitre 8. Contrôle et optimisation des E/S disque

Ce chapitre traite des performances des unités de disques locales.

Si vous êtes peu familier des concepts AIX de groupe de volumes, de volumes logiques et physiques, et de partitions physiques, reportez-vous à ”Gestion AIX du stockage sur disque fixe” : présentation des performances, page 2-13.

Cette rubrique traite également des points suivants :

- ”Optimisation des lectures séquentielles anticipées”
- ”Régulation des E/S disque”
- ”Performances et répartition des volumes logiques”
- ”Performances et taille de fragment du système de fichiers”
- ”Performances et compression”
- ”Performances et E/S disque asynchrones”
- ”Performances et E/S disque brutes”
- ”Performances et **sync**/**fsync**”
- ”Modification du paramètre **max_coalesce** du pilote SCSI”
- ”Limites de la file d’attente de l’unité de disque et de la carte SCSI”
- ”Contrôle du nombre de pbufs système”

Cette section traite des points suivants :

- ”Préinstallation”
- ”Élaboration d’une base d’optimisation”
- ”Évaluation des performances disque après installation”
- ”Évaluation du placement physique des données sur disque”
- ”Réorganisation d’un groupe ou d’un volume logique”
- ”Réorganisation d’un système de fichiers”
- ”Performances et espaces de pagination”
- ”Mesure des E/S disque globales via **vmstat**”
- ”Analyse détaillée des E/S via **filemon**”
- ”Programmes à disque limité”
- ”Extension de la configuration”

Préinstallation

La configuration des systèmes de fichiers a un impact non négligeable sur l’ensemble des performances du système et toute modification intervenant après l’installation prend beaucoup de temps. Décider du nombre et du type des disques fixes ainsi que de la taille et du placement des espaces de pagination et des volumes logiques sur ces disques est de ce fait une opération critique de la préinstallation.

Pour des précisions sur la planification de la configuration des disques au moment de la préinstallation, reportez-vous à ”Préinstallation du disque” page 4-24.

Élaboration d'une base d'optimisation

Avant toute modification de la configuration disque ou toute optimisation des paramètres, il est bon de définir une base de mesures, qui permette d'enregistrer la configuration et les performances actuelles. Outre vos propres mesures, vous pouvez créer une base exhaustive à l'aide du module PerfPMR. Reportez-vous à "Contrôle avant modification", page 2-14.

Évaluation des performances disque après installation

Commencez l'évaluation en lançant **iostat** avec un paramètre intervalle, pendant une pointe de charge ou l'exécution d'une application critique pour laquelle vous souhaitez minimiser les délais d'E/S. Le script suivant exécute **iostat** à l'arrière–plan tandis qu'un **cp** appliqué à un fichier volumineux est exécuté à l'avant–plan, de sorte qu'il y a quelques E/S à mesurer:

```
$ iostat 5 3 >io.out &
$ cp big1 /dev/null
```

Ce qui génère les trois relevés suivants dans `io.out`:

```
tty:     tin       tout    cpu:    % user    % sys    % idle    % iowait
         0.0        3.2              0.2      0.6     98.9        0.3


Disks:      % tm_act      Kbps       tps     msps    Kb_read    Kb_wrtn
hdisk0         0.0        0.3       0.0               29753      48076
hdisk1         0.1        0.1       0.0               11971      26460
hdisk2         0.2        0.8       0.1               91200     108355
cd0            0.0        0.0       0.0                   0          0
```

Le premier, récapitulatif, indique l'équilibre (ou ici le déséquilibre) global des E/S sur chaque disque. `hdisk1` est quasi inutilisé alors que `hdisk2` reçoit près de 63 % du total des E/S.

Le deuxième relevé indique l'intervalle de 5 secondes pendant lequel **cp** a été exécuté. Les données doivent être étudiées avec attention. La durée d'exécution de ce **cp** a été d'environ 2,6 secondes. Ainsi, 2,5 secondes de haute activité d'E/S ont été "compensées" par 2,5 secondes de temps inoccupé pour arriver aux 39,5 % `iowait` relevés. Un intervalle plus court aurait permis de caractériser plus précisément la commande elle-même, mais cet exemple illustre les précautions à prendre lors de l'examen de relevés indiquant des moyennes d'activité.

Évaluation du placement physique des données sur disque

Si la charge de travail se révèle très dépendante des E/S, vous avez intérêt à examiner la position physique des fichiers sur le disque pour décider s'il convient de les réorganiser. Pour étudier l'emplacement des partitions du volume logique `hd11` dans le volume physique `hdisk0`, utilisez :

```
$ lslv -p hdisk0 hd11
```

lslv génère :

```
hdisk0:hd11:/home/op
USED  USED  USED  USED  USED  USED  USED  USED  USED  USED     1-10
USED  USED  USED  USED  USED  USED  USED                     11-17

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED    18-27
USED  USED  USED  USED  USED  USED  USED                     28-34

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED    35-44
USED  USED  USED  USED  USED  USED                           45-50

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED    51-60
0052  0053  0054  0055  0056  0057  0058                     61-67

0059  0060  0061  0062  0063  0064  0065  0066  0067  0068    68-77
0069  0070  0071  0072  0073  0074  0075                     78-84
```

Le mot `USED` signifie que la partition physique est utilisée par un volume logique autre que `hd11`. Les chiffres indiquent la partition logique de `hd11` affectée à cette partition physique.

Pour consulter le reste de `hd11` sur `hdisk1`, entrez :

```
$ lslv -p hdisk1 hd11
```

qui génère :

```
hdisk1:hd11:/home/op
0035  0036  0037  0038  0039  0040  0041  0042  0043  0044    1-10
0045  0046  0047  0048  0049  0050  0051                    11-17

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED   18-27
USED  USED  USED  USED  USED  USED  USED                    28-34

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED   35-44
USED  USED  USED  USED  USED  USED                          45-50

0001  0002  0003  0004  0005  0006  0007  0008  0009  0010   51-60
0011  0012  0013  0014  0015  0016  0017                    61-67

0018  0019  0020  0021  0022  0023  0024  0025  0026  0027   68-77
0028  0029  0030  0031  0032  0033  0034                    78-84
```

Nous constatons que le volume logique `hd11` est réparti sur le volume physique `hdisk1`, ses premières partitions logiques se trouvant dans les zones médianes-internes et internes de `hdisk1`, et les partitions 35 à 51, dans les zones externes. Un travail qui accède à hd11 de façon aléatoire subit inutilement des attentes d'E/S, tandis que l'accesseur disque va et vient entre les différentes zones de hd11. Ces relevés montent également qu'il n'existe aucune partition physique libre ni sur `hdisk0` ni sur `hdisk1`.

Si nous examinons hd2 (volume logique contenant le système de fichiers */usr*) sur hdisk2 via :

```
$ lslv -p hdisk2 hd2
```

nous découvrons quelques partitions physiques libres (FREE) :

```
hdisk2:hd2:/usr
USED  USED  USED  USED  FREE  FREE  FREE  FREE  FREE  FREE     1-10
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    11-20
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    21-30
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    31-40
FREE                                                        41-41

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED    42-51
USED  USED  USED  USED  USED  USED  FREE  FREE  FREE  FREE    52-61
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    62-71
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    72-81
FREE                                                        82-82

USED  USED  0001  0002  0003  0004  0005  0006  0007  0008    83-92
0009  0010  0011  0012  0013  0014  0015  USED  USED  USED   93-102
USED  0016  0017  0018  0019  0020  0021  0022  0023  0024  103-112
0025  0026  0027  0028  0029  0030  0031  0032  0033  0034  113-122

0035  0036  0037  0038  0039  0040  0041  0042  0043  0044  123-132
0045  0046  0047  0048  0049  0050  0051  0052  0053  0054  133-142
0055  0056  0057  0058  0059  0060  0061  0062  0063  0064  143-152
0065  0066  0067  0068  0069  0070  0071  0072  0073  0074  153-162
0075                                                        163-163

0076  0077  0078  0079  0080  0081  0082  0083  0084  0085  164-173
0086  0087  0088  0089  0090  0091  0092  0093  0094  0095  174-183
0096  0097  0098  0099  0100  FREE  FREE  FREE  FREE  FREE  184-193
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  194-203
FREE                                                        204-204
```

Ce relevé présente quelques différences intéressantes par rapport aux précédents. Le volume logique hd2 est contigu, sauf pour quatre partitions physiques (100 à 103). D'autres **lslv** (non indiqués) montrent que ces partitions servent à hd1, hd3 et hd9var (**/home**, **/tmp** et **/var**, respectivement).

Pour voir comment le fichier copié précédemment, big1, est stocké sur disque, vous pouvez lancer la commande **fileplace** :

```
$ fileplace -pv big1
```

Le relevé correspondant est le suivant :

```
File: big1  Size: 3554273 bytes  Vol: /dev/hd10 (4096 byte blks)
Inode: 19  Mode: -rwxr-xr-x  Owner: frankw  Group: system

Physical blocks (mirror copy 1)                    Logical blocks
-------------------------------                    --------------
01584-01591  hdisk0       8 blks,    32 KB,   0,9%    01040-01047
01624-01671  hdisk0      48 blks,   192 KB,   5,5%    01080-01127
01728-02539  hdisk0     812 blks,  3248 KB,  93.5%    01184-01995

  956 blocks over space of 869:  space efficiency = 90,8%
  3 fragments out of 868 possible:  sequentiality = 99,8%
```

Ce qui indique qu'il y a très peu de fragmentation dans le fichier et que les "trous" sont petits. Nous pouvons en déduire que le mode de stockage de big1 a une incidence quasi nulle sur la durée de lecture séquentielle. Mieux, en supposant qu'un fichier de 3,5 Mo récemment créé subisse cette légère fragmentation, il apparaît que l'ensemble du système de fichiers est lui-même peu fragmenté.

Remarque : Si un fichier a été créé par recherche d'emplacements et écriture d'articles très dispersés, seules les pages contenant des articles occupent de la place sur le disque et apparaissent sur un relevé **fileplace**. Le système de fichiers *ne remplit pas*

automatiquement les pages concernées lorsque le fichier est créé. Si toutefois un fichier de ce type est lu séquentiellement, via la commande **cp** ou **tar**, par exemple, l'espace entre les articles est interprété comme des zéros binaires. Aussi, le résultat d'une commande **cp** peut-il être *bien plus grand* que le fichier d'entrée, bien que la quantité de données n'ait pas varié.

Sous AIX version 4.1, la commande **fileplace** est intégrée à la boîte à outils PTX (Performance Toolbox for AIX). Pour savoir si **fileplace** est disponible, entrez :

```
lslpp -lI perfagent.tools
```

Si le module est installé, **fileplace** est disponible.

Réorganisation d'un groupe ou d'un volume logique

Si vous découvrez un volume suffisamment fragmenté pour nécessiter une restructuration, vous pouvez faire appel à **smit** pour exécuter la commande **reorgvg** (**smit** –> **Mémoire physique et logique** –> **Gestionnaire de volumes logiques** –> **Groupes de volumes** –> **Définition caractéristiques d'un groupe de volumes** –> **Réorganisation d'un groupe de volumes**). Le raccourci est :

```
# smit reorgvg
```

Lancer cette commande sur `rootvg` sur le système test, sans spécifier de volumes logiques, entraîne la migration de tous les volumes logiques vers `hdisk2`. Après restructuration, le résultat de la commande :

```
$ lslv -p hdisk2 hd2
```

était :

```
hdisk2:hd2:/usr
USED  USED  USED  USED  USED  USED  USED  USED  FREE  FREE      1-10
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE     11-20
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE     21-30
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE     31-40
FREE                                                         41-41

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED     42-51
USED  USED  USED  USED  USED  USED  FREE  FREE  FREE  FREE     52-61
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE     62-71
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE     72-81
FREE                                                         82-82

USED  USED  0001  0002  0003  0004  0005  0006  0007  0008     83-92
0009  0010  0011  0012  0013  0014  0015  0016  0017  0018    93-102
0019  0020  0021  0022  0023  0024  0025  0026  0027  0028   103-112
0029  0030  0031  0032  0033  0034  0035  0036  0037  0038   113-122

0039  0040  0041  0042  0043  0044  0045  0046  0047  0048   123-132
0049  0050  0051  0052  0053  0054  0055  0056  0057  0058   133-142
0059  0060  0061  0062  0063  0064  0065  0066  0067  0068   143-152
0069  0070  0071  0072  0073  0074  0075  0076  0077  0078   153-162
0079                                                         163-163

0080  0081  0082  0083  0084  0085  0086  0087  0088  0089   164-173
0090  0091  0092  0093  0094  0095  0096  0097  0098  0099   174-183
0100  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE   184-193
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE   194-203
FREE                                                         204-204
```

La fragmentation des partitions physiques dans `hd2`, observée sur le relevé précédent, a disparu. Nous n'avons toutefois affecté *aucune* fragmentation au niveau du bloc physique existant éventuellement dans le système de fichiers /**usr**. Dans la mesure où la plupart des fichiers dans /**usr** sont écrits une fois, pendant l'installation du système, et ne sont plus mis à jour ensuite, /**usr** a peu de chance de subir des fragmentations. Ce qui n'est pas le cas des données utilisateur dans le système de fichiers /**home**.

Réorganisation d'un système de fichiers

Le système test comporte un volume logique et un système de fichiers, `hd11` (point de montage : `/home/op`), séparés, destinés aux tests risquant de détruire des données. Si vous décidez de réorganiser `hd11`, commencez par sauvegarder les données :

```
# cd /home/op
# find . -print | pax -wf/home/waters/test_bucket/backuptestfile
```

Ces commandes créent un fichier de sauvegarde (dans un système de fichiers distinct) contenant tous les fichiers du système à réorganiser. Si l'espace disque du système est limité, vous pouvez effectuer cette sauvegarde sur bande.

Pour reconstituer le système de fichiers, vous devez au préalable lancer la commande **unmount**, comme suit :

```
# unmount /home/op
```

Si des processus utilisent `/home/op` ou un de ses sous-répertoires, ils doivent être tués (via **kill**) pour que **unmount** aboutisse.

Pour reconstituer le système de fichiers sur le volume logique de `/home/op`, entrez :

```
# mkfs /dev/hd11
```

Une confirmation vous est demandée, avant destruction de l'ancien système de fichiers. Le nom du système de fichiers n'est pas modifié. Pour revenir à la situation antérieure (`/home/op` étant néanmoins vidé), entrez :

```
# mount /dev/hd11 /home/op
# cd /home/op
```

Restaurez ensuite les données :

```
# pax -rf/home/frankw/tuning.io/backuptestfile >/dev/null
```

La sortie standard est réacheminée vers `/dev/null` pour éviter l'affichage de tous le noms de fichiers, affichage qui peut demander beaucoup de temps.

Si vous examinez à nouveau le fichier volumineux examiné précédemment, via :

```
# fileplace -piv big1
```

vous constatez qu'il est à présent presque contigu :

```
File: big1  Size: 3554273 bytes  Vol: /dev/hd11 (4096 byte blks)
Inode: 8290  Mode: -rwxr-xr-x  Owner: frankw  Group: system

INDIRECT BLOCK: 60307

Physical blocks (mirror copy 1)                    Logical blocks
-------------------------------                    --------------
60299-60306  hdisk1     8 blks,    32 KB,   0,9%    08555-08562
60308-61167  hdisk1   860 blks,  3440 KB,  99.1%    08564-09423

  868 blocks over space of 869:  space efficiency = 99.9%
  2 fragments out of 868 possible:  sequentiality = 99.9%
```

L'option **–i** spécifiée avec la commande **fileplace** montre que le trou de 1 bloc, entre les huit premiers blocs du fichier et le reste, contient le bloc indirect, requis pour compléter les informations d'i-node lorsque la longueur du fichier dépasse huit blocs.

Performances et espaces de pagination

Les E/S de et vers les espaces de pagination sont aléatoires, effectuées le plus souvent une page à la fois. Les relevés **vmstat** indiquent la quantité d'espace de pagination occupé par les E/S. Les deux exemples suivants montrent l'activité de pagination au cours d'une compilation C, sur une machine artificiellement restreinte via **rmss**. Les colonnes **pi** et **po** (pages chargées et pages déchargées de l'espace de pagination) indiquent les E/S dans l'espace de pagination (exprimées en pages de 4096 octets) au cours de chaque intervalle de 5 secondes. Le premier relevé, récapitulatif, a été supprimé. Notez que l'activité de pagination se produit par rafales.

```
$ vmstat 5
procs     memory              page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b    avm    fre  re  pi  po  fr   sr  cy  in    sy  cs us sy id wa
 0  0   2502    432   0   0   0   0    0   0 134    26  20  0  1 99  0
 0  0   2904    201   4   0   7  43 1524   0 129   227  38 64 12 15 10
 1  0   3043    136   0   0   0  17  136   0 117    46  24 92  6  0  2
 1  0   3019     90   3   0   0   0    0   0 126    74  34 84  6  0 10
 0  0   3049    178   2   0  15  28  876   0 148    32  32 85  6  0  9
 1  0   3057    216   0   1   6  11   77   0 121    39  25 93  5  0  2
 0  0   2502    599   2  15   0   0    0   0 142 1195  69 47  9 11 34
 0  0   2502    596   0   0   0   0    0   0 135    30  22  1  1 98  1
```

Les relevés "avant" et "après" **vmstat –s** suivants indiquent les cumuls d'activités de pagination. N'oubliez pas que ce sont les valeurs de "`paging space page ins`" et "`. . outs`" qui représentent l'activité d'E/S réelle. Les décomptes (non qualifiés) de "`page ins`" et de "`page outs`" indiquent le total des E/S (dans l'espace de pagination, mais aussi E/S normales de fichiers également traitées par le mécanisme de pagination). (Les informations ne relevant pas de notre propos ont été supprimées des relevés.)

```
$ vmstat -s                          $ vmstat -s
        .                                    .
     6602 page ins                        7022 page ins
     3948 page outs                       4146 page outs
      544 paging space page ins            689 paging space page ins
     1923 paging space page outs          2032 paging space page outs
       71 total reclaims                    84 total reclaims
        .                                    .
        .                                    .
```

Le fait qu'il y ait eu plus de pages déchargées que de pages chargées au cours de la compilation laisse supposer que le système a été restreint à la limite de son point d'emballement. Certaines pages ont été référencées deux fois car leur trame a été volée avant la fin de leur utilisation (c'est-à-dire avant toute modification).

Mesure des E/S disque globales via vmstat

La technique décrite ci-dessus peut également être utilisée pour évaluer la charge d'E/S disque générée par un programme. Si le système est inoccupé par ailleurs, la séquence :

```
$ vmstat -s >statout
$ testpgm
$ sync
$ vmstat -s >> statout
$ egrep "ins|outs" statout
```

génère un relevé avant et après, indiquant les cumuls d'activités disque, tels que :

```
     5698 page ins
     5012 page outs
        0 paging space page ins
       32 paging space page outs
     6671 page ins
     5268 page outs
        8 paging space page ins
      225 paging space page outs
```

Au cours de la période d'exécution de la commande (compilation d'un vaste programme C), le système a lu un total de 981 pages (dont 8 issues de l'espace de pagination) et écrit un total de 449 pages (dont 193 dans l'espace de pagination).

Analyse détaillée des E/S via filemon

La commande **filemon** fait appel à l'utilitaire de suivi pour générer une image détaillée de l'activité d'E/S au cours d'un intervalle défini. Dans la mesure où la commande **filemon** invoque cet utilitaire, seul l'utilisateur `root` ou un membre du groupe `system` est habilité à la lancer.

Sous AIX version 4.1, la commande **filemon** est intégrée à la boîte à outils PTX (Performance Toolbox for AIX). Pour déterminer si **filemon** est disponible, entrez :

```
lslpp -lI perfagent.tools
```

Si ce module est installé, **filemon** est disponible.

Le suivi, lancé par la commande **filemon**, peut être suspendu par **trcoff,** relancé par **trcon** et arrêté par **trcstop**. Dès que le suivi est terminé, **filemon** en envoie le compte rendu à **stdout**. La séquence suivante illustre l'utilisation de **filemon** :

```
# filemon -o fm.test.out ; cp smit.log /dev/null ; trcstop
```

Voici le relevé généré par cette séquence (sur un système inoccupé par ailleurs) :

```
Wed Jan 12 11:28:25 1994
System: AIX alborz Node: 3 Machine: 000249573100

0.303 secs in measured interval
Cpu utilization:  55.3%

Most Active Segments
------------------------------------------------------------------------
  #MBs  #rpgs  #wpgs  segid  segtype              volume:inode
------------------------------------------------------------------------
   0.1     26      0   0984  persistent           /dev/hd1:25
   0.0      1      0   34ba  .indirect            /dev/hd1:4

Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.66    216      0  357.0  /dev/hd1             /home

Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.65    216      0  357.0  /dev/hdisk1          320  MB SCSI

------------------------------------------------------------------------
Detailed VM Segment Stats   (4096 byte pages)
------------------------------------------------------------------------

SEGMENT: 0984  segtype: persistent  volume: /dev/hd1  inode: 25
segment flags:          pers
reads:                  26      (0 errs)
  read times (msec):    avg  45.644 min   9.115 max 101.388 sdev  33.045
  read sequences:       3
  read seq. lengths:    avg     8.7 min       1 max      22 sdev     9.5
SEGMENT: 34ba  segtype: .indirect  volume: /dev/hd1  inode: 4
segment flags:          pers jnld sys
reads:                  1       (0 errs)
  read times (msec):    avg  16.375 min  16.375 max  16.375 sdev   0.000
  read sequences:       1
  read seq. lengths:    avg     1.0 min       1 max       1 sdev     0.0

------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------
```

```
VOLUME: /dev/hd1  description: /home
reads:                  27      (0 errs)
  read sizes (blks):    avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):    avg  44.316 min   8.907 max 101.112 sdev  32.893
  read sequences:       12
  read seq. lengths:    avg    18.0 min       8 max      64 sdev    15.4
seeks:                  12      (44.4%)
  seek dist (blks):     init    512
                        avg   312.0 min       8 max    1760 sdev   494.9
time to next req(msec): avg   8.085 min   0.012 max  64.877 sdev  17.383
throughput:             357.0 KB/sec
utilization:            0.66
-------------------------------------------------------------------------
Detailed Physical Volume Stats   (512 byte blocks)
-------------------------------------------------------------------------

VOLUME: /dev/hdisk1  description: 320  MB SCSI
reads:                  14      (0 errs)
  read sizes (blks):    avg    15.4 min       8 max      32 sdev     8.3
  read times (msec):    avg  13.989 min   5.667 max  25.369 sdev   5.608
  read sequences:       12
  read seq. lengths:    avg    18.0 min       8 max      64 sdev    15.4
seeks:                  12      (85.7%)
  seek dist (blks):     init 263168,
                        avg   312.0 min       8 max    1760 sdev   494.9
  seek dist (cyls):     init    399
                        avg     0.5 min       0 max       2 sdev     0.8
time to next req(msec): avg  27.302 min   3.313 max  64.856 sdev  22.295
throughput:             357.0 KB/sec
utilization:            0.65
```

Le relevé `Most Active Segments` donne la liste des fichiers les plus actifs. Pour identifier les fichiers inconnus, vous pouvez traduire le nom du volume logique, /**dev**/**hd1**, vers le point de montage du système de fichiers, /**home**, et lancer la commande **find** :

```
# find /home -inum 25 -print
```

qui renvoie :

```
/home/waters/smit.log
```

Lancer **filemon** sur des systèmes traitant de véritables travaux génère des relevés bien plus longs, avec l'éventualité de devoir agrandir l'espace du tampon de suivi.
La consommation d'espace et de temps CPU par **filemon** peuvent entraîner une sérieuse dégradation des performances : testez **filemon** sur un système non productif avant de vous aventurer sur un système réellement exploité.

Remarque : Bien que **filemon** indique la moyenne, le minimum, le maximum et la déviation standard dans ses rubriques relatives aux statistiques, ces résultats ne peuvent servir de base à la définition d'intervalles fiables ou à d'autres conclusions statistiques formelles. La répartition des points de données n'est en général ni aléatoire ni symétrique.

Programmes à disque limité

Les problèmes liés au disque sont de différents ordres, avec des solutions diversifiées :

- Si de vastes travaux en arrière-plan, très consommateurs d'E/S, interfèrent avec les temps de réponses interactives, vous pouvez activer la régulation d'E/S.
- S'il apparaît qu'un petit nombre de fichiers sont lus et relus à répétition, envisagez d'ajouter de la mémoire réelle pour que la mise en tampon de ces fichiers soit plus efficace.
- Si **iostat** indique que les activités d'E/S sont inégalement réparties sur les unités de disque et que l'une de ces unités accuse une charge dépassant 70 à 80 %, envisagez de réorganiser le système de fichiers.

- Si les accès sont essentiellement de type aléatoire, envisagez d'ajouter des disques et de mieux répartir les fichiers concernés.
- Si les accès effectués par le travail sont majoritairement séquentiels et impliquent plusieurs unités de disque, envisagez d'ajouter une ou plusieurs cartes disque. Il peut être intéressant de construire un volume logique réparti, pour les fichiers séquentiels volumineux dont les performances sont critiques.

La section suivante donne plus de précisions sur les ratios unités/cartes disque.

Extension de la configuration

Malheureusement, toute opération d'optimisation des performances a des limites. La question devient alors : "De quel matériel ai–je besoin, en quelle quantité et comment en tirer le maximum ?" La question est particulièrement épineuse avec des travaux limités au niveau des disques, du fait du nombre de variables. Pour améliorer les performances de ce type de travaux, vous pouvez :

- Ajouter des unités de disque et y répartir les données : la charge des E/S est ainsi divisée entre davantage d'accesseurs.
- Acquérir des unités de disque plus rapides pour compléter ou remplacer les unités existantes, pour les données très sollicitées.
- Ajouter une ou plusieurs cartes disque SCSI pour connecter les unités nouvelles et/ou existantes.
- Ajouter de la RAM au système et augmenter les valeurs des paramètres VMM **minperm** et **maxperm** pour améliorer la mise en mémoire cache des données très sollicitées.

Vu la complexité de la question, étroitement liée à l'importance de la charge de travail et de la configuration, et vu la rapidité d'évolution de la vitesse des disques, des cartes et des processeurs, nous ne vous donnerons pas ici de consignes précises, mais de simples conseils de base.

- Si vous cherchez à optimiser les accès séquentiels :
  - Ne raccordez pas plus de trois unités (nouvelles) de 1 Go à une carte disque SCSI–2 donnée.

  Les performances maximales d'une carte disque SCSI–2, en traitement séquentiel continu, dans des conditions idéales, sont d'environ 6,8 Mo/s.
- Si vous cherchez à optimiser les accès séquentiels :
  - Ne raccordez pas plus de six unités (nouvelles) de 1 Go à une carte disque SCSI–2 donnée.

  Les performances maximales d'une carte disque SCSI–2, en traitement aléatoire (sur des pages de 4 ko) continu, dans des conditions idéales, sont d'environ 435 pages/s.

Pour obtenir une analyse plus fine de votre configuration et de votre charge de travail, vous pouvez utiliser un simulateur, tel BEST/1.

Voir aussi

Commandes **backup**, **fileplace**, **lslv**, **lsps**, **lspv**, **reorgvg**, **smit** et **unmount**.

"Performances du gestionnaire de mémoire virtuelle (VMM)"

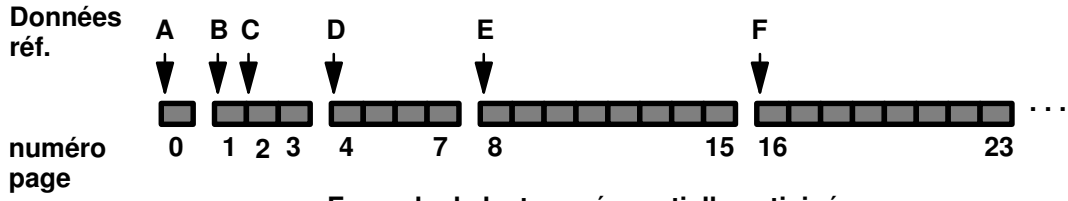
"Programmes à CPU limitée"

"Position et taille des espaces de pagination"

Optimisation des lectures séquentielles anticipées

La fonction VMM de lecture séquentielle anticipée, décrite à ”Lecture séquentielle anticipée”, peut améliorer les performances des programmes accédant séquentiellement à des fichiers volumineux.

Les cas où l’activation de cette fonction améliore les performances sont rares. Néanmoins, l’analyste des performances doit comprendre l’interaction entre cette fonction et l’application ainsi qu’avec les autres paramètres d’optimisation des E/S disque. La figure ”Exemple de lecture séquentielle anticipée” illustre une situation classique.



Exemple de lecture séquentielle anticipée

Dans cet exemple, **minpgahead** a la valeur 2 et **maxpgahead**, la valeur 8 (valeurs par défaut). Le programme traite le fichier séquentiellement. Seules les références des données significatives pour la fonction de lecture anticipée sont illustrées (A à F). Les étapes sont les suivantes :

A Le premier accès au fichier provoque la lecture de la première page (page 0) de ce fichier. A ce niveau, VMM ne fait aucune hypothèse sur la nature de l’accès (séquentiel ou aléatoire).

B Lorsque le programme accède au premier octet de la page suivante (page 1), sans autre accès aux autres pages du fichier, VMM conclut à un accès séquentiel. Il programme la lecture de **minpgahead** (2) pages (pages 2 et 3). L’accès B entraîne ainsi la lecture de 3 pages.

C Lorsque le programme accède au premier octet de la première page lue par anticipation (page 2), VMM double la valeur du nombre de pages à lire par anticipation (qui passe à 4) et programme la lecture des pages 4 à 7.

D Lorsque le programme accède au premier octet de la première page lue par anticipation (page 4), VMM double la valeur du nombre de pages à lire par anticipation (qui passe à 8) et programme la lecture des pages 8 à 15.

E Lorsque le programme accède au premier octet de la première page lue par anticipation (page 8), VMM détermine que le nombre de pages à lire par anticipation a atteint la valeur de **maxpgahead** et programme la lecture des pages 16 à 23.

F VMM poursuit la lecture de **maxpgahead** pages lorsque le programme accède au premier octet du groupe précédent des pages à lire par anticipation, jusqu’à la fin du fichier.

(Si le programme passe en cours de route à un autre type d’accès, la fonction de lecture anticipée est arrêtée. Elle est relancée avec **minpgahead** pages si VMM détecte une reprise des accès séquentiels par le programme.)

Les paramètres **minpgahead** et **maxpgahead** peuvent être modifiés via la commande **vmtune**. Si vous envisagez de le faire, n’oubliez pas que :

- Les valeurs doivent appartenir à l’ensemble : 0, 1, 2, 4, 8, 16. Toute autre valeur peut avoir l’effet inverse de celui escompté ou des effets fonctionnels non souhaités.
  - Les valeurs doivent être des puissances de 2, du fait de l’algorithme de doublement.
  - Les valeurs de **maxpgahead** supérieures à 16 (lecture anticipée de plus de 64 ko) excèdent la capacité de certains pilotes de disque.

  - Des valeurs supérieures de **maxpgahead** peuvent être utilisées sur des systèmes où les performances en mode séquentiel des volumes logiques répartis sont d'une importance capitale.

- Donner à **minpgahead** la valeur 0 annihile de fait le mécanisme. Ce qui peut avoir de fâcheuses conséquences sur les performances.
- La valeur par défaut (8) de **maxpgahead** est celle qui entraîne les performances maximales des E/S séquentielles pour les unités de disque courantes.
- Le passage de la valeur de lecture anticipée de **minpgahead** à **maxpgahead** est suffisamment rapide : pour les fichiers de taille courante, il n'y a aucun intérêt à augmenter la valeur de **minpgahead**.

Régulation des E/S disque

La régulation des E/S disque a pour but d'empêcher les programmes générant beaucoup de sorties de saturer les fonctions d'E/S du système et de ralentir les temps de réponse des programmes peu demandeurs. La régulation des E/S force les niveaux haut et bas par segment (c'est-à-dire, en fait, par fichier) à la somme des E/S en attente. Lorsqu'un processus tente d'écrire dans un fichier dont le niveau d'écriture en attente est au plus haut, le processus est mis en veille jusqu'à ce que le nombre d'E/S en attente redevienne inférieur ou égal au niveau bas. La logique de traitement des E/S reste identique. Seules les sorties issues de processus générateurs de volumes élevés sont quelque peu ralenties. Les niveaux haut et bas sont définis via **smit** par les options **Environnements système et processus** –> **Modif/affich caractéristiques du système d'exploitation** (et entrée du nombre de pages correspondant aux deux niveaux). Leur valeur par défaut est de 0, ce qui désactive la régulation. Les nouvelles valeurs des paramètres de régulation prennent normalement effet immédiatement et restent en vigueur jusqu'à nouvelle modification.

Exemple

L'effet de la régulation sur les performances peut être étudié en lançant une session **vi** sur un nouveau fichier, tandis qu'un autre processus copie (via **cp**) un fichier de 64 Mo. La copie a lieu du `disk1` vers le `disk0 et` l'exécutable **vi** réside sur le `disk0`**.** Pour que la session **vi** démarre, elle doit se charger elle–même et effectuer quelques autres E/S, ce qu'elle effectue en mode aléatoire, une page à la fois. Cette opération mobilise environ 50 E/S physiques, qui peuvent être terminées en 0,71 secondes, sous réserve qu'il n'y ait de concurrence pour l'accès au disque. Si le niveau haut est défini à 0 (valeur par défaut), les écritures logiques résultant de **cp** passent avant les écritures physiques et une longue file d'attente se construit. Chaque E/S initiée par **vi** doit attendre son tour dans la file avant que l'E/S suivante puisse être émise et **vi** ne peut ainsi pas démarrer avant la fin de **cp**. La figure "Résultats d'un test de régulation d'E/S" illustre les délais d'exécution de **cp** et d'initialisation de **vi** avec différents paramètres de régulation. Cette étude a été effectuée sur un Modèle 530 équipé de deux disques de 857 Mo et de 32 Mo de mémoire vive.

Niveau haut	Niveau bas	cp (s)	vi (s)
0	0	50	**vi** non fait
0	0	50,2	**vi** terminé après **cp** est terminé
9	6	76.8	2.7
17	12	57.9	3.6
17	8	63.9	3.4
33	24	52.0	9.0
33	16	55.1	4.9

Résultats d'un test de régulation d'E/S

Notez que la durée de **cp** est toujours plus élevée lorsque la régulation est activée. La régulation sacrifie quelques sorties de programmes très consommateurs d'E/S pour améliorer les temps de réponse d'autres programmes. La difficulté pour un administrateur est de trouver un équilibre débit/temps de réponse qui réponde aux priorités de l'entreprise.

Les niveaux haut et bas ont été choisis par tâtonnements et erreurs, sur la base de notre connaissance du chemin des E/S. Ce choix n'est pas simple du fait de la combinaison des écritures différées et asynchrones. Les niveaux hauts de la forme 4x + 1 sont particulièrement efficaces, car :

- La fonction d'écriture différée envoie sur disque les 4 pages précédentes lorsqu'une écriture logique a lieu sur le premier octet de la cinquième page.

- Si le niveau haut de régulation était un multiple de 4 (disons 8), un processus l'atteindrait lorsqu'il demande une écriture qui s'étend jusqu'à la 9ème page. Il serait alors mis en veille – *avant* que l'algorithme d'écriture différée ait la moindre chance de détecter que la quatrième page utilisée est terminée et que les quatre pages sont prêtes à être écrites.
- Le processus sera alors maintenu en veille avec quatre pages pleines prêtes à sortir, jusqu'à ce que le niveau de ses écritures passe en dessous du niveau bas de régulation.
- Si, par ailleurs, le niveau haut a été défini à 9, l'écriture différée va avoir à programmer les quatre pages pour la sortie avant que le processus ne soit suspendu.

Une des limites de la régulation intervient lorsqu'un processus écrit dans des tampons de taille supérieure à 4 ko, auquel cas le contrôle possible baisse singulièrement.
Si, lorsqu'une écriture est envoyée à VMM, le niveau haut n'a pas été atteint, VMM exécute un lancement des E/S sur toutes les pages du tampon, même s'il en résulte un dépassement du niveau haut. La régulation fonctionne bien sur **cp** car cette commande écrit par tranches de 4 ko, mais si **cp** écrivait dans des tampons plus grands, les délais illustrés à la figure "Résultats du test de régulation des E/S" pour le lancement de **vi** seraient supérieurs.

Niveau haut	Niveau bas	cp (s)	vi (s)
0	0	50	**vi** non fait
0	0	50,2	**vi** terminé après **cp** est terminé
9	6	76.8	2.7
17	12	57.9	3.6
17	8	63.9	3.4
33	24	52.0	9.0
33	16	55.1	4.9

Résultats d'un test de régulation d'E/S

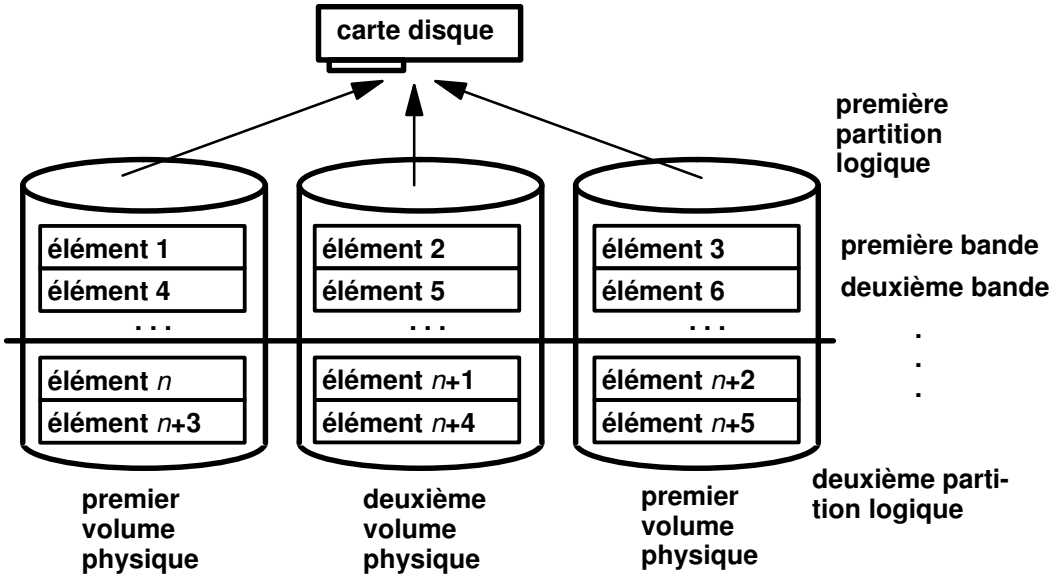
La régulation des E/S disque est un paramètre d'optimisation qui peut améliorer les temps de réponse interactifs dans le cas où des programmes (en avant ou arrière-plan) qui écrivent d'importants volumes interfèrent avec des requêtes en avant-plan. Mal utilisée, elle peut à l'inverse réduire le débit de façon excessive. Les paramètres de la figure Résultats du test de régulation des E/S constituent une bonne base de départ, mais seule l'expérience vous aidera à déterminer les paramètres adaptés à votre charge de travail.

Parmi les programmes susceptibles d'imposer d'activer la régulation des E/S, citons :

- Les programmes générant par algorithmes de forts volumes de sorties et ne subissant de ce fait aucune contrainte quant à la durée de la lecture : une régulation peut être nécessaire sur des processeurs rapides et pas sur des processeurs lents.
- Les programmes écrivant de longs fichiers peu modifiés, peu de temps après leur lecture intégrale (par une commande antérieure, par exemple).
- Les filtres, telle la commande **tar**, qui lisent un fichier et le réécrivent après un traitement minimal. Le besoin de régulation est d'autant plus pressant si l'entrée est lue à partir d'une unité de disque plus rapide que l'unité sur laquelle est écrite la sortie.

Performances et répartition des volumes logiques

La répartition en bandes est une technique qui consiste à distribuer les données d'un volume logique sur plusieurs unités de disque, de sorte que la capacité d'E/S des disques puisse être exploitée en parallèle pour l'accès aux données du volume logique. (La fonction de création de volumes logiques répartis en bandes n'est pas disponible sous la version 3.2.5.) Le but principal de l'opération est d'accroître les performances au niveau de la lecture et de l'écriture de longs fichiers séquentiels. La figure "Volume logique réparti /dev/lvs0" en donne un exemple.



Volume logique réparti /dev/lvs0

Sur un volume logique classique, les adresses des données correspondent à la séquence de blocs de la partition physique sous-jacente. Sur un volume logique réparti en bandes, ces adresses suivent la séquence des éléments de bande. Une bande complète est constituée d'un élément par unité physique contenant une partie du volume logique réparti. Le LVM détermine les blocs et les unités physiques correspondant au bloc en cours de lecture ou d'écriture. Si plusieurs disques sont concernés, les E/S requises sont programmées simultanément pour toutes les unités.

Par exemple, supposons que **lvs0** est constitué d'éléments de bande de 64 ko, est composé de six partitions de 2 Mo et contient un système de fichiers journalisé (JFS). Si une application est en cours de lecture d'un fichier séquentiel long et que la lecture anticipée a atteint un point de stabilité, chaque lecture va entraîner la programmation de deux ou trois E/S vers chaque unité de disque, pour un total de huit pages (en supposant que le fichier se trouve sur des blocs consécutifs du volume logique). Les opérations de lecture sont exécutées dans l'ordre déterminé par le pilote de l'unité de disque. Les données requises sont reconstituées à partir des différents éléments des entrées et retournées à l'application.

Bien que chaque unité de disque ait un délai de latence initial différent, fonction de la position du mécanisme d'accès au début de l'opération, une fois que le processus a atteint un état stable, les trois disques procèdent aux lectures à une vitesse proche de leur vitesse maximale.

Conception d’un volume logique réparti

Pour définir un volume logique réparti, vous spécifiez :

unités	A l’évidence, il faut disposer d’au moins deux disques physiques. Les unités choisies doivent en outre être peu chargées d’autres activités au moment des E/S séquentielles critiques. Certaines combinaisons de carte/unité de disque nécessitent de diviser la charge de travail d’un volume logique réparti entre deux ou plusieurs cartes.
taille d’élément de bande	Bien qu’en théorie toute puissance de 2 comprise entre 4 et 128 ko convienne, vous devez tenir compte des lectures anticipées, puisque la plupart des lectures passeront par ce mécanisme. L’objectif est que chaque lecture anticipée génère au moins une E/S sur chaque unité de disque (idéalement un nombre égal).
taille	Le nombre de partitions physiques affectées au volume logique doit être un multiple du nombre de disques utilisés.
attributs	Ne peuvent être mis en miroir, c’est-à-dire que copies = 1.

Optimisation des E/S d’un volume logique réparti

Dans des situations d’évaluation, les techniques suivantes ont généré les plus forts débits d’E/S séquentielles :

- Taille d’élément de bande de 64 ko.
- **max_coalesce** de 64 ko (valeur par défaut). Égal à la taille d’un élément de bande.
- **minpgahead** de 2.
- **maxpgahead** de 16 fois le nombre d’unités de disque. Ceci entraîne la lecture anticipée de page par unités de la taille d’un élément de bande (64 ko) multiplié par le nombre d’unités de disque. Chaque lecture anticipée provoque donc la lecture d’un élément de bande de chaque unité de disque.
- Requêtes d’E/S pour (64 ko fois le nombre d’unités de disque). Egal à la valeur de **maxpgahead**.
- Modification de **maxfree** en fonction de celle de **maxpgahead**. Reportez–vous à ”Paramètres minfree et maxfree”, page 7-16.
- Tampons d’E/S alignés sur 64 octets. Si le volume logique doit occuper des unités physiques connectées à au moins deux cartes disque, les tampons d’E/S utilisés doivent être affectés dans la limite de 64 octets. Ceci pour éviter que le LVM ne sérialise les E/S sur les différents disques. Le code suivant génère un pointeur de tampon aligné sur 64 octets :

```
char *buffer;
buffer = malloc(MAXBLKSIZE+64);
buffer = ((int)buffer + 64) & ~0x3f;
```

Si les volume logiques répartis se trouvent sur des volume logiques bruts et que des écritures de plus de 1,125 Mo sont effectuées sur ces volumes logiques bruts répartis, augmenter le paramètre **lvm_bufcnt** de **vmtune** peut augmenter le débit des écritures.

Performances et taille de fragment du système de fichiers

La fonction de *fragmentation* (AIX version 4.1 seulement) permet d'affecter l'espace d'un système de fichiers par tranches inférieures à 4 ko. A la création d'un système de fichiers, l'administrateur système peut spécifier la taille de ces fragments : 512, 1024, 2048 et 4096 octets (valeur par défaut). Les fichiers plus petits qu'un fragment sont stockés dans un seul fragment, économisant ainsi de l'espace, ce qui est le but premier de l'opération.

Les fichiers de taille inférieure à 4096 octets sont stockés dans le minimum nécessaire de fragments contigus. Les fichiers de taille comprise entre 4096 octets et 32 ko (inclus) sont stockés dans un ou plusieurs blocs entiers (4 ko) et dans autant de fragments que requis pour contenir le reste. Les fichiers de plus de 32 ko sont stockés entièrement dans des blocs complets.

Quelle que soit la taille d'un fragment, un bloc complet est supposé contenir 4096 octets. Si, toutefois, la taille de fragment d'un système de fichiers est inférieure à 4096, un bloc peut être constitué d'une séquence quelconque de fragments contigus totalisant 4096 octets. Il ne doit pas nécessairement commencer à la limite d'un multiple de 4096 octets.

Le système de fichiers tente d'affecter aux fichiers des fragments contigus. Dans cette optique, il répartit les fichiers dans le volume logique pour éviter les interférences d'affectation inter-fichiers et la fragmentation.

Le principal écueil qui guette les systèmes de fichiers dotés d'une taille de fragment minime est la fragmentation de l'espace. La présence de petits fichiers disséminés au sein du volume logique rend difficile, voire impossible, l'affectation de blocs contigus à ou même simplement proches d'un fichier quelque peu volumineux. L'accès à ce type de fichiers devient moins performant. Poussée à l'extrême, la fragmentation de l'espace peut rendre impossible l'affectation d'espace à un fichier, même s'il existe suffisamment de fragments isolés libres.

Une part de la décision de créer un système de fichier avec de petits fragments doit être une stratégie de défragmentation de l'espace dans ce système de fichiers via la commande **defragfs**. Cette stratégie doit également tenir compte du coût de performance induit par l'exécution de **defragfs**.

Performances et compression

Lorsqu'un fichier est écrit dans un système de fichiers pour lequel la compression est active, l'algorithme correspondant comprime les données par tranches de 4 096 octets (une page) et écrit les données compressées dans le minimum de fragments contigus. A l'évidence, si la taille de fragment du système de fichiers est de 4 ko, il n'y a pas de gain d'espace disque compensant l'effort de compression des données. (La compression et les fragments de taille inférieure à 4 ko sont une nouveauté de AIX version 4.1.)

Bien que la compression génère un gain d'espace, il convient toutefois de ménager un peu d'espace libre dans le système de fichiers.

- En effet, dans la mesure où le degré de compression de chaque bloc de 4096 octets n'est pas connu à l'avance, le système de fichiers commence par réserver un bloc entier. Les fragments inutilisés sont libérés après la compression, mais la réservation initiale peut entraîner un message de type "manque d'espace" prématuré.
- Il faut de l'espace pour l'exécution de la commande **defragfs**.

Performances et E/S disque asynchrones

Les applications peuvent appeler les sous-routines **aio_read** et **aio_write** pour effectuer des E/S disque asynchrones. Le contrôle revient à l'application dès que la requête est mise en file d'attente. Celle-ci peut alors continuer le traitement pendant l'exécution des opérations sur disque.

Bien que l'application puisse poursuivre le traitement, un processus noyau (kproc) appelé *serveur* prend en charge chaque requête depuis le moment où elle est extraite de la file d'attente jusqu'à la fin de son exécution. Le nombre de serveurs limite le nombre d'E/S disque asynchrones exécutables simultanément. Le nombre de serveurs est défini via **smit** (**smit**–>Unités–>E-S asynchrones–>Modif/affich caractéristiques d'E-S asynchrones–>{MINIMUM|MAXIMUM} nombre de serveurs ou **smit aio**) ou via **chdev**. Le nombre minimal de serveurs est le nombre de serveurs à lancer à l'amorçage du système. Le maximum limite le nombre de serveurs qui peuvent être démarrés en réponse à un grand nombre de requêtes simultanées.

Les valeurs par défaut sont **minservers=1** et **maxservers=10**. Sur les systèmes où peu d'applications font appel à des E/S asynchrones, ces valeurs conviennent généralement. Dans un environnement riche en unités de disque et en applications clés qui utilisent des E/S asynchrones, ces valeurs par défaut sont largement insuffisantes. Le déficit de serveurs entraîne un ralentissement apparent des E/S disque. Non seulement le temps passé par les requêtes dans la file d'attente est démesuré, mais la faiblesse du ratio serveurs/unités de disque signifie que les algorithmes d'optimisation des recherches disposent de trop peu de requêtes à exploiter pour chaque unité de disque.

Dans les environnements où les performances au niveau des E/S disque asynchrones sont capitales et le volume des requêtes élevé, nous vous conseillons de :

- donner à **maxservers** une valeur au moins égale à 10*(nombre de disques dont l'accès est asynchrone)
- donner à **minservers** la valeur de **maxservers**/2.

Pour un système équipé de 3 disques dont l'accès est asynchrone, vous pouvez à cet effet lancer la commande :

```
# chdev -l aio0 -a minservers='15' -a maxservers='30'
```

Remarque : Les actions AIO effectuées sur un volume logique RAW ne font pas appel aux processus serveur kproc. En l'occurrence, les paramètres "maxservers" et "minservers" n'ont aucun effet.

Performances et E/S disque brutes

Un programme dispose de trois moyens pour accéder à un disque en mode brut :

- Les fichiers spéciaux d'unités disque brutes par bloc sont dotés de noms de la forme /**dev**/**hdisk***n* et sont utilisés par certains sous-systèmes. Ces unités ne doivent pas être exploitées par les programmes d'application.
- Les fichiers spéciaux d'unités disque brutes par caractère sont dotés de noms de la forme /**dev**/**rhdisk***n*. L'exploitation de ces unités par les programmes d'application est déconseillée. Si vous décidez néanmoins d'adopter cette technique, vérifiez qu'aucun volume logique AIX n'occupe une partie de l'unité de disque physique en cours d'accès. L'effet sur les performances de l'interaction entre accès brut et accès du type système de fichiers à la même unité physique est imprévisible. Assurez–vous de ne pas écraser les 512 premiers octets du disque puisque c'est là qu'est enregistré l'ID du volume physique.
- Il est possible d'accéder en mode brut à un volume logique sur lequel aucun système de fichiers n'a été créé. Toutes les opérations **write**, **read**, **lseek**, etc., doivent être effectuées sur des multiples de 512 octets. Violer cette règle a pour conséquence, entre autres, une très nette dégradation des performances.

Performances et sync/fsync

La synchronisation forcée du contenu de la mémoire réelle et des disques s'opère de différentes façons :

- Un programme d'application effectue un appel **fsync()** pour un fichier donné : toutes les pages contenant des données modifiées pour ce fichier sont alors écrites sur disque. L'écriture prend fin lorsque l'appel **fsync()** revient au programme.
- Un programme d'application effectue un appel **sync()** : toutes les pages du fichier en mémoire contenant des données modifiées sont alors programmées pour une écriture sur disque. L'écriture ne prend pas *nécessairement* fin lorsque l'appel **sync()** revient au programme.
- Un utilisateur lance la commande **sync**, laquelle à son tour émet un appel **sync()**. Là aussi, certaines écritures peuvent ne pas être terminées lorsque l'utilisateur est invité à entrer une autre commande (ou que la commande suivante d'un script shell est traitée).
- Le démon sync, /**usr**/**sbin**/**syncd**, lance un appel **sync()** à intervalle régulier – généralement de 60 secondes. Ceci garantit que le système n'accumule pas de grandes quantités de données existant uniquement en RAM volatile.

Outre une faible consommation de CPU, une synchronisation :

- regroupe les écritures au lieu de les éparpiller ;
- écrit au moins 28 ko de données système, même en l'absence d'activité d'E/S depuis la dernière synchronisation ;
- accélère l'écriture des données sur disque, en annulant l'algorithme d'écriture différée. Cet effet est particulièrement visible sur les programmes qui émettent un **fsync()** après chaque écriture.

Modification du paramètre max_coalesce du pilote SCSI

Si la file d'attente de l'unité de disque SCSI contient un grand nombre de requêtes d'E/S disque, l'unité tente de les fusionner pour en diminuer le nombre. La taille maximale de requête (en termes de données transmises) que l'unité SCSI est autorisée à constituer est définie par le paramètre **max_coalesce**. Normalement, **max_coalesce** a la valeur 64 ko.

Pour exploiter au maximum les volumes logiques répartis et les piles de disques, il peut être souhaitable d'augmenter la valeur de **max_coalesce**. Pour ce faire, une strophe de la base de données ODM PdAt doit spécifier la nouvelle valeur de **max_coalesce**. Si vous avez déjà inséré une strophe de ce type, vous pouvez en obtenir la version courante via :

```
# odmget -q \
"uniquetype=disk/scsi/osdisk AND attribute=max_coalesce" \
PdAt > foo
```

S'il n'existe aucune strophe de ce type, créez, via un éditeur, le fichier `foo` contenant :

```
PdAt:
        uniquetype = "disk/scsi/osdisk"
        attribute = "max_coalesce"
        deflt = "0x20000"
        values = "0x20000"
        width = ""
        type = "R"
        generic = ""
        rep = "n"
        nls_index = 0
```

Notez que **max_coalesce**, en octets, est exprimé par un nombre hexadécimal. La valeur des champs `deflt` et `values` (0x20000) donne à **max_coalesce** la valeur de 128 ko. Remplacez ensuite l'ancienne strophe de PdAt (le cas échéant) par `foo`, comme suit :

```
# odmdelete -o PdAt \
-q "uniquetype=/disk/scsi/osdisk AND attribute=max_coalesce"
# odmadd < foo
```

Pour appliquer la modification, reconstituez le noyau et réamorcez le système, via :

```
# bosboot -a -d hdisk0
# shutdown -rF
```

Limites de la file d'attente de l'unité de disque et de la carte SCSI

AIX a la possibilité de forcer les limites du nombre de requêtes d'E/S issues de la carte SCSI vers un bus ou une unité de disque SCSI donné. Ces limites ont pour but d'exploiter la capacité du matériel à gérer des requêtes multiples, tout en assurant que les algorithmes d'optimisation de la recherche des pilotes d'unités peuvent effectivement opérer.

Pour les unités non BULL, il est parfois judicieux de modifier les valeurs par défaut des limites de la file d'attente AIX, choisies à l'origine pour faire face aux situations les moins favorables. Les sections suivantes traitent des cas où il convient de modifier ces valeurs.

Unité de disque non BULL

Pour les unités de disque BULL, le nombre par défaut de requêtes pouvant être émises simultanément est de 3. Cette valeur est basée sur des considérations de performance complexes et aucune interface directe n'est fournie pour la modifier. La longueur par défaut d'une file d'attente matérielle des unités de disque non BULL est de 1. Si une unité non BULL spécifique n'a pas la capacité de mettre en tampon plusieurs requêtes, la description système de cette unité doit être modifiée en conséquence.

Par exemple, voici les caractéristiques par défaut d'une unité de disque non BULL affichées par la commande **lsattr** :

```
$ lsattr -D -c disk -s scsi -t osdisk
pvid          none Physical volume identifier      False
clr_q         no   Device CLEARS its Queue on error
q_err         yes  Use QERR bit
q_type        none Queuing TYPE
queue_depth   1    Queue DEPTH
reassign_to   120  REASSIGN time out value
rw_timeout    30   READ/WRITE time out value
start_timeout 60   START unit time out value
```

Pour modifier ces paramètres, vous pouvez passer par **smit** (raccourci **chgdsk**) et par la commande **chdev**. Par exemple, si votre système contient l'unité de disque SCSI non BULL **hdisk5**, la commande :

```
# chdev -l hdisk5 -a q_type=simple -a queue_depth=3
```

active la fonction de mise en file d'attente pour cette unité et définit à 3 la longueur de cette file d'attente.

Pile de disques non BULL

Une pile de disques est considérée par AIX comme un seul disque, assez volumineux. Une pile de disques non BULL, comme une unité de disque non BULL, est de la classe **disk**, sous-classe **scsi**, type **osdisk** (c'est–à–dire "Other SCSI Disk Drive"). Une pile de disques contenant en réalité un certain nombre de disques physiques, chacun prenant en charge de multiples requêtes, la longueur de la file d'attente pour la pile de disques doit être suffisamment élevée pour permettre une exploitation efficace de toutes les unités physiques. Par exemple, si **hdisk7** est une pile de huit disques non BULL, effectuez la modification comme suit :

```
# chdev -l hdisk7 -a q_type=simple -a queue_depth=24
```

Si la pile de disques est connectée via un bus SCSI–2 Fast/Wide, il faudra peut-être modifier également la limite définie pour les requêtes externes pour ce bus.

Limitation des requêtes externes sur une carte disque

La carte SCSI–2 Fast/Wide accepte deux bus SCSI : l'un pour les unités internes, l'autre pour les unités externes. Une limite sur le nombre total de requêtes en attente est définie pour chaque bus (valeur par défaut : 40, valeur maximale 128). Si une pile de disques BULL est connectée à un bus de carte SCSI–2 Fast/Wide, la limite pour le bus est augmentée pour tenir compte de la taille de la file d'attente de la pile de disques. S'il s'agit d'une pile de disques non BULL, la limite pour le bus doit être définie manuellement. Par exemple, pour limiter à 70 le nombre de requêtes externes de la carte **scsi3**, spécifiez :

```
# chdev -l scsi3 -a num_cmd_elems=70
```

Sur le contrôleur haute performance SCSI–2, le nombre de requêtes en file d'attente est limité à 30 et cette limite ne peut être modifée. De ce fait, vous devez vérifier que la somme des longueurs des files d'attente reliées à un contrôleur de ce type ne dépasse pas 30.

La carte SCSI ESCALA d'origine ne prend pas en charge la mise en file d'attente. Ne connectez pas d'unité de piles de disques à une carte de ce type.

Contrôle du nombre de pbufs système

Le gestionnaire LVM (Logical Volume Manager) utilise un élément appelé ”pbuf” pour contrôler les E/S disque en attente. Sous AIX version 3, il est requis un pbuf par page lue ou écrite. Sur les systèmes effectuant beaucoup d’E/S séquentielles, le pool de pbufs peut se trouver rapidement dégarni. La commande **vmtune** permet d’augmenter le nombre de pbufs pour compenser cette déperdition.

Sous AIX version 4.1, un seul pbuf est utilisé par requête d’E/S séquentielle, quel que soit le nombre de pages en cause. Ce qui diminue grandement la probabilité de manquer de pbufs.

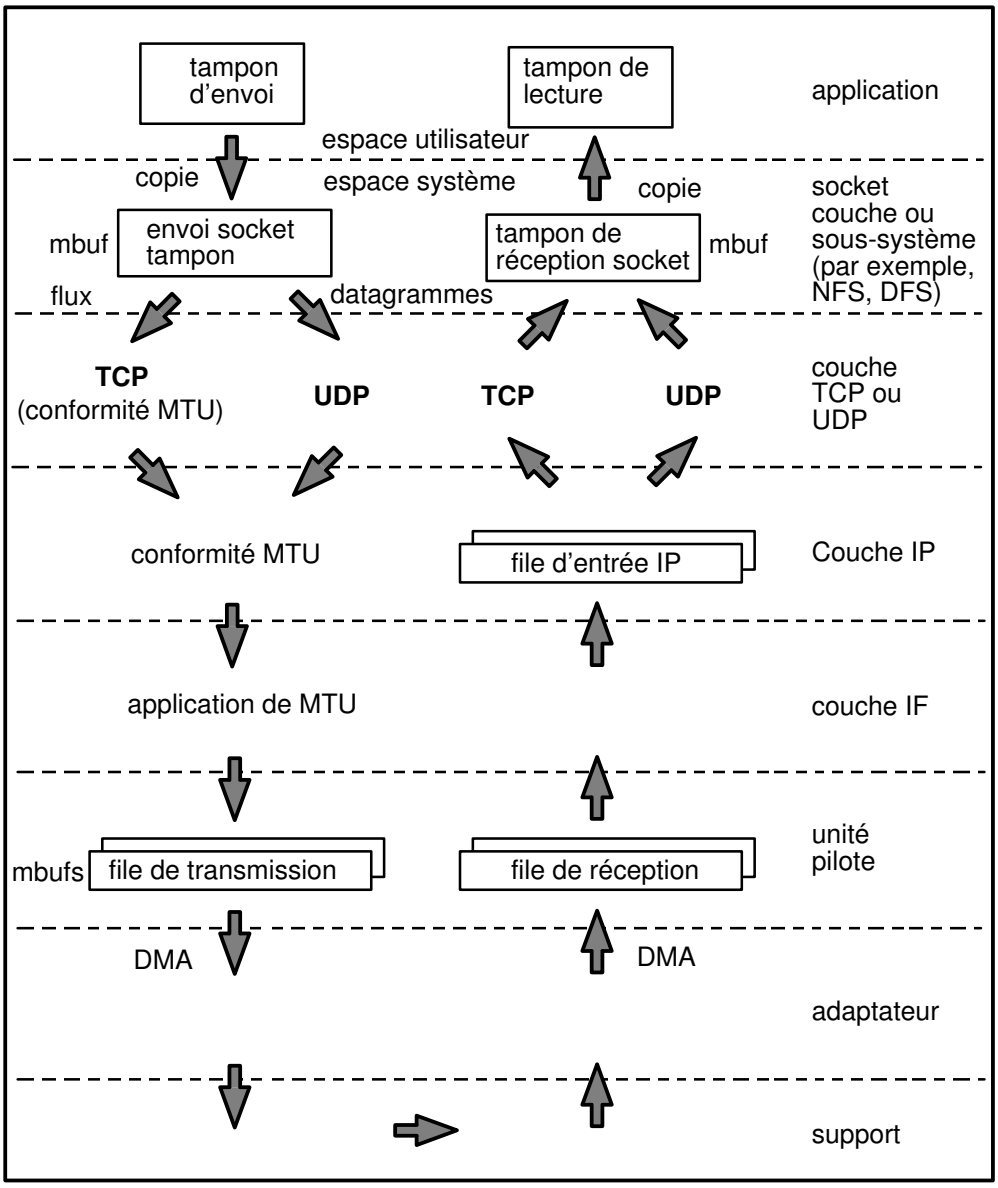
Chapitre 9. Contrôle et optimisation des E/S de communication

Ce chapitre traite de différents protocoles de communication, et notamment des moyens de les contrôler et de les optimiser. Il comporte les sections suivantes :

- ”Performances UDP/TCP/IP”
- ”Optimisation de TCP et UDP”
- ”Récapitulatif des paramètres d’optimisation UDP, TCP/IP et mbuf”
- ”Optimisation de NFS”
- ”Service des stations de travail sans disque”
- ”Optimisation des connexions asynchrones pour transferts haut débit”
- ”Evaluation des performances réseau avec **netpmon**”
- ”Analyse des problèmes de performance avec **iptrace**”

Performances UDP/TCP/IP

Pour appréhender les caractéristiques des performances de UDP et de TCP/IP, quelques notions sur l'architecture sous-jacente s'imposent. La figure "Flux des données UDP/TCP/IP" illustre la structure traitée ici.



Flux des données UDP/TCP/IP

Cette figure illustre le cheminement des données depuis une application sur un système à une autre sur un système distant. Le traitement des couches, détaillé plus loin, peut être résumé comme suit (en ne tenant pas compte du traitement des erreurs et des limites des tampons) :

- La requête d'écriture émanant de l'application provoque la copie des données du segment de travail de l'application dans le tampon d'envoi du socket.
- La couche ou le sous-système socket transmet les données à UDP ou à TCP.
- Si les données sont plus volumineuses que le MTU (maximum transfer unit) du réseau :
  - TCP fragmente la sortie en segments conformes à la limite MTU.

  - UDP délègue la fragmentation de la sortie à la couche IP.
- Au besoin, IP fragmente la sortie en éléments compatibles avec le MTU.
- La couche IF (interface) vérifie qu'aucun paquet sortant ne dépasse la limite MTU.
- Les paquets sont placés dans la file de sortie de l'unité et transmis par l'adaptateur de réseau au système destinataire.
- A l'arrivée, les paquets sont placés dans la file de réception du pilote d'unité, et transmis à IP via la couche Interface.
- Si l'IP du système destinataire détermine que l'IP du système expéditeur a fragmenté un bloc de données, il rassemble les fragments dans leur forme d'origine et passe les données à TCP ou à UDP :
  - TCP réassemble les segments d'origine et place l'entrée dans le tampon de réception du socket.
  - UDP passe simplement l'entrée dans le tampon de réception du socket.
- Lorsque l'application émet une requête de lecture, les données ad hoc sont copiées depuis le tampon de réception du socket (se trouvant dans la mémoire du noyau) dans le segment de travail de l'application.

Gestion de la mémoire du sous-système de communication (mbuf)

Pour prévenir la fragmentation de la mémoire du noyau et une surcharge d'appels à **xmalloc()**, les pools de tampons usuels sont partagés par les différentes couches du sous-système de communication. La fonction de gestion des mbuf contrôle deux pools de tampons : un pool de tampons de faible capacité (256 octets chacun), simplement appelés *mbuf*, et un pool de tampons de capacité élevée (4 096 octets chacun), généralement appelés *grappes mbuf* ou *grappes.* Les pools sont constitués d'éléments fixes de mémoire virtuelle du noyau – ce qui signifie qu'ils résident en permanence en mémoire physique et ne font jamais partie de pages évacuées. En conséquence, la capacité de mémoire réelle, disponible pour la pagination dans les programmes d'application et les données, est diminuée d'un volume équivalent à l'augmentation de la capacité des pools mbuf.

Outre le fait d'éviter la duplication, le partage des pools mbuf et grappes permet aux différentes couches de se passer les pointeurs, réduisant les appels de gestion mbuf et les copies de données.

Couche socket

Les sockets fournissent l'interface de programmation API au sous-système de communication. Il en existe plusieurs types, offrant différents niveaux de service par le biais de protocoles de communication différenciés. Les sockets de type SOCK_DGRAM utilisent le protocole UDP. Les sockets de type SOCK_STREAM utilisent le protocole TCP.

La sémantique d'ouverture, de lecture et d'écriture relative aux sockets est semblable à celle qui régit la manipulation des fichiers.

La taille des tampons de la mémoire virtuelle du système (c'est-à-dire la capacité totale des pools mbuf) utilisés en entrée et sortie des sockets est limitée par les valeurs par défaut du système (il est possible de passer outre ces valeurs pour un socket donné par un appel à la routine **setsockopt()**) :

udp_sendspace et **udp_recvspace**	Taille des tampons des sockets de datagrammes. Les valeurs par défaut sont respectivement de 9216 et 41600.
tcp_sendspace et **tcp_recvspace**	Taille des tampons des sockets de flux. Par défaut : 16384 (les deux).

Pour afficher ces valeurs, entrez :

```
$ no -a
```

Pour les définir (en tant qu'utilisateur `root`), entrez, par exemple :

```
# no -o udp_sendspace=nouvelle-valeur
```

Nouvelle-valeur doit être inférieure ou égale au paramètre **sb_max**, qui contrôle la taille maximale d'espace utilisable par un tampon d'envoi ou de réception de socket. **sb_max** est affiché par **no –a** et défini (*avant* toute tentative d'augmentation de sa valeur courante) par la commande **no** :

```
# no -o sb_max=nouvelle-limite
```

Remarque : La taille des tampons d'envoi et de réception de socket ne doit pas dépasser **sb_max** octets, **sb_max** constituant une valeur plafond au niveau de la consommation d'espace tampon. Les deux quantités ne sont toutefois pas mesurées de la même manière : la taille du tampon limite la quantité de *données* qui peut s'y trouver ; **sb_max** limite le nombre d'*octets de mbuf* qui peut se trouver dans le tampon à un instant donné. Dans un environnement Ethernet par exemple, chaque grappe mbuf de 4 096 octets peut contenir juste 1500 octets de données. Dans ce cas, **sb_max** doit être 2,73 fois supérieur à la taille de tampon de socket, pour atteindre la capacité spécifiée. Empiriquement, nous préconisons de donner à **sb_max** une valeur au moins double de la taille du tampon de socket.

Flux d'envoi

Lorsqu'une application écrit dans un socket, les données sont copiées de l'espace utilisateur dans le tampon d'envoi du socket, dans l'espace du noyau. Selon la quantité de données copiées, le socket les place dans le mbuf ou dans les grappes. Une fois la copie terminée, la couche socket appelle la *couche transport* (TCP ou UDP), et lui passe un pointeur sur la liste des mbuf liés (*chaîne mbuf*).

Flux de réception

Du côté réception, une application ouvre un socket et y lit des données. Si le tampon de réception n'en contient pas, la couche socket met la routine de l'application à l'état "veille" (bloqué) jusqu'à l'arrivée de données. Lorsque des données arrivent, elles sont placées dans la file du tampon de réception du socket et la routine de l'application peut être diffusée. Les données sont ensuite copiées dans le tampon de l'application (dans l'espace utilisateur), la chaîne mbuf est libérée et le contrôle revient à l'application.

Création de socket

Sous AIX Version 4.3.1 et ultérieures, la valeur **sockthresh** détermine la proportion de la mémoire réseau du système qui peut être utilisée avant que la création de socket ne soit désactivée. La valeur de **sockthresh** est fournie sous la forme d'un pourcentage de **thewall**. Elle est par défaut de 85% et peut prendre toute valeur comprise entre 1 et 100, à condition toutefois que **sockthresh** ne soit pas inférieure à la quantité de mémoire actuellement utilisée.

L'option **sockthresh** a pour but d'éviter que de trop nombreuses connexions soient ouvertes, conduisant à l'utilisation de la totalité de la mémoire réseau disponible sur la machine. Dans ce cas de figure, il ne resterait plus de mémoire pour les autres opérations et la machine, bloquée, devrait être redémarrée. Utilisez **sockthresh** pour définir à partir de quel point les nouveaux sockets sont refusés. Les appels vers **socket()** et **socketpair()** échoueront (erreur ENOBUFS), et les requêtes de connexion entrantes seront annulées de manière transparente. Ainsi, la mémoire réseau restant disponible sur la machine pourra être utilisée par les connexions existantes et aucun blocage de la machine ne sera à craindre.

Le décompte **netstat –m** `sockets not created because sockthresh was reached` est incrémenté chaque fois que la création d'un nouveau socket est refusé parce que la quantité de mémoire réseau déjà utilisée est supérieure à la valeur de **sockthresh**.

Pour afficher **sockthresh**, entrez :

```
$ no -o sockthresh
```

Pour le définir (en tant qu'utilisateur root), entrez :

```
# no -o sockthresh=Nouvelle-valeur
```

Pour affecter à **sockthresh** sa valeur par défaut, entrez :

```
# no -d sockthresh
```

Fonctions UDP et TCP

Les deux sections suivantes décrivent les fonctions UDP et TCP. Pour faciliter la comparaison entre les deux, chaque section est divisée en rubriques : connexion, détection des erreurs, reprise sur erreur, contrôle du flux, taille des données et gestion du MTU.

Couche UDP

UDP offre un protocole économique pour les applications dotées de fonctions susceptibles de traiter les incidents de communication. UDP convient bien aux applications de style "requête-réponse". Une application de ce type devant de toutes façons gérer les incidents de réponse, il suffit de peu pour lui faire gérer les erreurs de communication comme une des causes possibles d'un défaut de réponse. C'est pourquoi, outre son faible coût, des sous-systèmes tels NFS, ONC RPC, DCE RPC et DFS utilisent UDP.

Connexion	Aucun. UDP est un protocole sans état. Chaque requête reçue de l'appelant est gérée indépendamment de celles qui la précèdent ou la suivent. (Si la sous-routine **connect()** est appelée pour un socket de datagramme, les informations sur la destination sont considérées comme une technique pour placer en mémoire cache l'adresse résolue, pour un usage ultérieur. Il n'y a pas de liaison effective entre le socket et l'adresse, ou d'incidence sur l'UDP du système destinataire.)
Détection d'erreur	Création et vérification de total de contrôle. L'UDP expéditeur établit le total de contrôle et l'UDP destinataire le vérifie. Si le contrôle est négatif, le paquet est abandonné.
Recouvrement d'erreur	Aucun. UDP n'accuse pas réception des paquets et ne détecte pas leur perte – en cours de transmission ou par saturation du pool de tampons. En conséquence, UDP ne retransmet jamais de paquet. La reprise doit être exécutée par l'application.
Contrôle de flux	Aucun. Lorsqu'UDP est invité à émettre, il envoie un paquet à IP. Lorsqu'un paquet arrive en provenance d'IP, il est placé dans le tampon de réception du socket. Si la file du tampon de la carte/du pilote d'unité est saturée, le paquet est abandonné sans aucune notification d'erreur. C'est à l'application ou au sous-système expéditeur de détecter l'erreur (délai imparti dépassé) et de relancer la transmission.
Taille des données	Doit tenir dans un seul tampon. Ce qui implique que les pools de tampons de deux côtés de UDP disposent de tampons dont la taille réponde aux besoins des applications. La taille maximale d'un paquet UDP est de 64 ko. Bien entendu, une application qui génère des blocs plus grands peut les scinder elle-même en plusieurs datagrammes – ce que fait DCE, par exemple – mais il est plus simple de passer par TCP.
Gestion de MTU	Aucun. Le traitement des données de taille supérieure à la taille du MTU (maximum transfer unit) pour l'interface est laissé à la charge d'IP. Si IP doit fragmenter les données pour les adapter au MTU, la perte des fragments devient une erreur, qui doit être gérée par l'application ou le sous-système.

Flux d'envoi

Si **udp_sendspace** est suffisamment grand pour contenir le datagramme, les données de l'application sont copiées dans des mbuf de la mémoire du noyau. Si le datagramme est plus grand que **udp_sendspace**, une erreur est renvoyée à l'application.

Si le datagramme est supérieur ou égal à 936 octets, il est copié dans une ou plusieurs grappes de 4 ko. Le reste (et l'éventuel datagramme complet), de moins de 936 octets est copié dans 1–4 mbufs. Par exemple, une écriture de 8704 octets est copiée dans deux grappes et le reste dans trois mbuf. UDP ajoute l'en-tête UDP (dans le même mbuf, si possible), effectue un total de contrôle sur les données et appelle la routine IP **ip_output**.

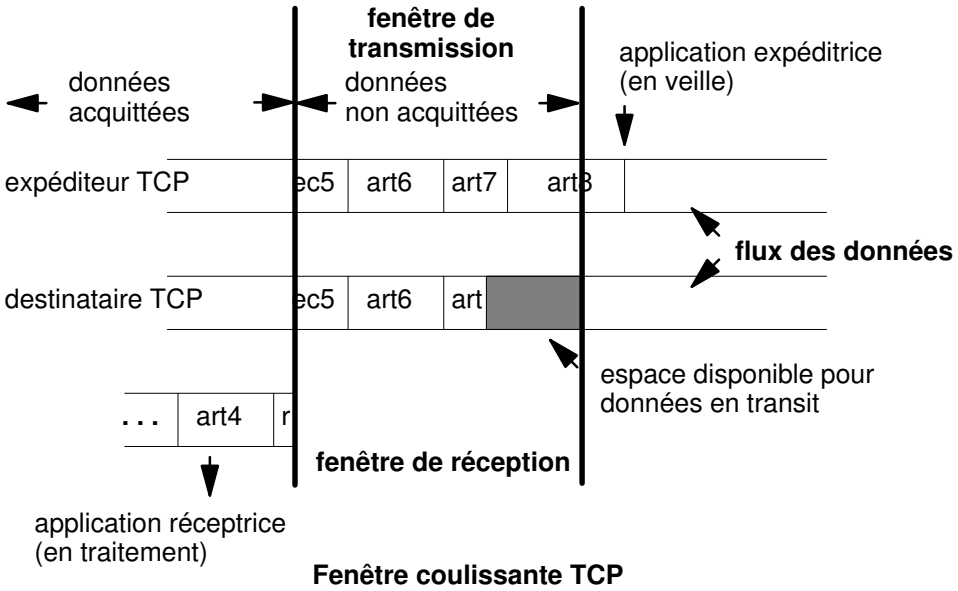
Flux de réception

UDP vérifie le total de contrôle et met les données en file d'attente sur le socket adéquat. Si la limite **udp_recvspace** est dépassée, le paquet est rejeté. (Le décompte de ces rejets est consigné par **netstat –s** sous l'intitulé "`udp:`" "`socket buffer overflows`.")
Si l'application est en attente d'un **receive** ou d'un **read** sur le socket, elle est placée en file d'attente d'exécution. **receive** copie alors le datagramme dans l'espace d'adressage de l'utilisateur et libère le mbuf : l'opération **receive** est terminée. Normalement, le destinataire adresse un accusé de réception et un message à l'expéditeur.

Couche TCP

TCP fournit un protocole de transmission fiable. TCP convient aux applications qui, au moins par périodes, sont essentiellement en mode entrée ou en mode sortie. TCP assurant que les paquets arrivent à destination, l'application est libérée de la détection d'erreur et des tâches de reprise. Parmi les applications exploitant le transport via TCP, citons : **ftp**, **rcp** et **telnet**. DCE peut l'utiliser s'il est configuré pour exploiter un protocole orienté connexion.

Connexion	Explicite. L'instance de TCP recevant la requête de connexion d'une application (dite l'*initiateur*) établit une session avec son homologue sur l'autre système (dit l'*écouteur).* Tous les échanges de données et de paquets de contrôle ont lieu dans le cadre de cette session.
Détection d'erreur	Création et vérification de total de contrôle. Le TCP expéditeur établit le total de contrôle et le TCP destinataire le vérifie. Si le contrôle est négatif, le destinataire ne reçoit pas d'accusé de réception du paquet.
Recouvrement d'erreur	Intégrale. TCP détecte les erreurs au niveau des totaux de contrôle et les pertes de paquet ou de fragment, via le dépassement de délai. En cas d'erreur, TCP retransmet les données jusqu'à ce que la réception aboutisse (ou notifie l'application d'une erreur irrémédiable).
Contrôle de flux	Renforcée. TCP adopte une technique dite de fenêtre coulissante pour vérifier la livraison à l'application destinataire. Le concept de fenêtre coulissante est illustré à la figure "Fenêtre coulissante TCP". (Les enregistrements illustrés ont pour seul objet de clarifier les choses. TCP traite les données comme un flux d'octets et ne garde pas trace des limites d'articles, qui sont définies par l'application).



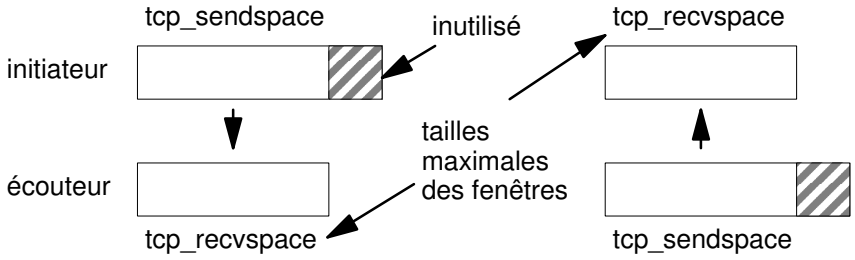
Fenêtre coulissante TCP

Sur la figure, l'application expéditrice est en veille, car elle a tenté d'écrire des données qui auraient conduit TCP à saturer le tampon du socket d'envoi (**tcp_sendspace**). Le TCP expéditeur détient toujours la fin de art5, l'intégralité de art6 et art7, et le début de art8. Le TCP destinataire n'a pas encore reçu la fin de art7, ni art8. L'application réceptrice a reçu art4 et le début de art5 lors de sa dernière lecture du socket, et elle procède actuellement au traitement des données. Lorsqu'elle effectuera une nouvelle lecture du socket, elle recevra (en supposant une lecture suffisamment vaste) la fin de art5, art6, et les parties de art7 et art8 arrivées à ce moment.

Au moment de cette lecture, le TCP destinataire acquitte ces données, le TCP expéditeur les abandonne, l'écriture en cours se termine et l'application expéditrice passe à l'état actif. (Pour éviter de surcharger le trafic réseau lorsque l'application lit de faibles quantités de données, TCP diffère l'acquittement jusqu'à ce que l'application réceptrice ait lu une quantité de données au moins égale à la moitié de la taille de la fenêtre réceptrice ou au double de la taille maximale de segment.)

Au cours de l'établissement d'une session, l'initiateur et l'écouteur dialoguent pour déterminer leur capacité respective de mise en tampon des données en entrée et en sortie. La taille de la fenêtre est celle de la plus petite des tailles. A mesure que les données sont inscrites dans le socket, elles sont placées dans le tampon de l'expéditeur. Lorsque le destinataire indique qu'il dispose d'espace, l'expéditeur transmet suffisamment de données pour remplir cet espace (s'il en dispose, bien entendu). Lorsque l'application réceptrice lit le socket, le TCP récepteur renvoie autant de données qu'il en détient dans son tampon. Il informe ensuite l'expéditeur que les données ont bien été transmises. Ce n'est qu'alors que l'expéditeur supprime les données de son propre tampon, déplaçant effectivement la fenêtre vers la droite, en proportion des données transmises. Si la fenêtre est saturée, parce que l'application réceptrice ne suit pas, la routine expéditrice est bloquée (ou reçoit un **errno** spécifique) lorsqu'elle tente d'écrire dans le socket.

La figure "Taille des fenêtres TCP" illustre la relation entre la taille des tampons du socket et celle de la fenêtre.



Taille des fenêtres TCP

tcp_recvspace est, dans les deux exemples, inférieur à **tcp_sendspace** : dans la mesure où la technique de déplacement de fenêtre impose que les deux systèmes aient la même capacité de tampon disponible, la taille de la fenêtre est définie à son minimum dans les deux directions. L'espace nominal supplémentaire disponible illustré n'est jamais utilisé.

Si le paramètre **rfc1323** est à 1, la taille maximale de la fenêtre TCP est de 4 Go (et non de 64 ko).

Taille des données	Indéfinie. TCP ne traite ni articles ni blocs, mais un flux continu d'octets. Si la capacité d'un tampon d'envoi excède celle que peut traiter le tampon de réception, le flux d'octets est segmenté en paquets de la taille du MTU. Dans la mesure où il gère l'insuffisance d'espace tampon de façon transparente, TCP ne garantit pas que le nombre et la taille des données reçues soient identiques à celles envoyées. C'est à l'application d'identifier (des deux côtés), le cas échéant, les limites des blocs ou des articles dans le flux de données. **Remarque :** Pour échanger des messages de type requête/réponse via TCP, l'application doit passer par **setsockopt** pour activer l'option TCP_NODELAY. Ce qui provoque l'envoi immédiat du message par TCP (dans le cadre des contraintes de la fenêtre coulissante), même s'il n'atteint pas la taille MTU. Sinon, TCP attend jusqu'à 200 millisecondes d'autres données avant de transmettre le message. Ce qui induit une baisse des performances.
Gestion de MTU	Géré par segmentation dans TCP. Lors de l'établissement de la connexion, l'initiateur et l'écouteur négocient la taille de segment maximale (MSS) à utiliser. Le MSS est normalement inférieur au MTU (voir "Optimisation de la taille maximale de segment (MSS) TCP", page 9-21). Si le paquet de sortie dépasse le MSS, TCP effectue la segmentation, rendant inutile la fragmentation dans IP. Le TCP récepteur place normalement les segments dans la file de réception du socket à mesure de leur arrivée. Si le TCP récepteur détecte la perte d'un segment, il refuse l'acquittement et renvoie les segments suivants tant qu'il ne reçoit pas le segment manquant.

Il n'existe bien entendu rien qui ressemble à une fonction libre. Les autres opérations exécutées par TCP pour assurer la fiabilité de la connexion induisent un surcoût au niveau processeur d'environ 7 à 12 % par rapport à UDP.

Flux d'envoi

Lorsque la couche TCP reçoit une requête d'écriture de la couche socket, elle affecte un nouveau mbuf pour les informations d'en-tête et copie les données dans le tampon d'envoi du socket, dans le mbuf d'en tête TCP (s'il y a de la place), ou dans une nouvelle chaîne mbuf. Si les données copiées se trouvent dans des grappes, de nouvelles grappes ne sont pas créées : un champ pointeur est défini dans le nouvel en-tête mbuf (cet en-tête fait partie de la structure mbuf et n'a aucune relation avec l'en-tête TCP), pointant sur les grappes contenant les données, évitant ainsi la charge d'une ou plusieurs copies de 4 ko. TCP effectue ensuite un total de contrôle sur les données, met à jour les diverses variables d'état utilisées pour le contrôle de flux et autres services, et, enfin, appelle la couche IP avec le mbuf d'en-tête désormais lié à la nouvelle chaîne mbuf.

Flux de réception

Lorsque la routine d'entrée TCP reçoit des données d'IP, elle effectue un total de contrôle sur l'en-tête et les données TCP, pour détecter d'éventuelles altérations, détermine la connexion à laquelle sont destinées les données, supprime les informations d'en-tête, lie la chaîne mbuf au tampon de réception du socket associé à la connexion, et fait appel à un service socket pour activer l'application (si elle est en veille, comme expliqué plus haut).

Couche IP

Le protocole Internet (IP) fournit un service datagramme de base aux couches hautes. S'il reçoit un paquet plus grand que le MTU de l'interface, il le fragmente et envoie les fragments au système destinataire, lequel les réassemble sous leur forme d'origine. En cas de perte d'un fragment au cours de la transmission, le paquet incomplet est rejeté par le destinataire. Le délai d'attente par IP d'un fragment manquant est défini par le paramètre **ipfragttl**, affiché et renseigné via la commande **no**.

La taille maximale de la file d'attente de paquets IP reçus de l'interface réseau est contrôlée par le paramètre **ipqmaxlen**, défini et affiché via **no**. Si la taille de la file d'attente en entrée atteint ce chiffre, les paquets suivants sont abandonnés.

Flux d'envoi

Lorsque la routine de sortie IP reçoit un paquet émis par UDP ou TCP, elle identifie l'interface à laquelle envoyer la chaîne mbuf, met à jour et effectue un total de contrôle sur la partie IP de l'en-tête, et passe le paquet à la couche interface (IF).

IP détermine le pilote et la carte d'unité à utiliser, en fonction du numéro du réseau. La table d'interface du pilote définit le MTU maximal pour ce réseau. Si le datagramme est plus petit que le MTU, IP insère l'en-tête IP dans le mbuf existant, effectue un total de contrôle sur l'en-tête IP et appelle le pilote pour envoyer la trame. Si la file d'envoi du pilote est saturée, une erreur EAGAIN est renvoyée à IP, qui la transmet à UDP, lequel la transmet à son tour à l'application. L'expéditeur n'a plus qu'à patienter et réessayer.

Si le datagramme est plus grand que le MTU (ce qui ne se produit que dans UDP), IP le divise en fragments de taille égale à celle du MTU, ajoute à chaque fragment un en-tête IP (dans un mbuf), et appelle le pilote pour chaque trame de fragment. Si la file d'envoi du pilote est saturée, une erreur EAGAIN est renvoyée à IP. IP rejette tous les fragments non envoyés associés à ce datagramme et renvoie EAGAIN à UDP, qui le transmet à l'application expéditrice. UDP renvoie EAGAIN à l'application expéditrice. IP et UDP ne plaçant pas les messages en file d'attente, il appartient à l'application de différer l'envoi et de le tenter ultérieurement.

Flux de réception

Sous AIX version 3, lorsque la routine d'entrée IP reçoit le contrôle suite à une interruption hors niveau planifiée par IF, elle ôte la chaîne mbuf de la file d'attente, vérifie le total de contrôle de l'en-tête IP pour s'assurer qu'il n'est pas altéré, et détermine si le paquet est destiné à ce système. Dans l'affirmative, et si la trame n'est pas un fragment, IP passe la chaîne mbuf à la routine d'entrée TCP ou UDP.

Sous AIX version 4.1, la couche demux (démultiplexeur – couche IF dans la version 3) appelle IP sur la routine d'interruption. Toute activité de planification ou de mise (ou retrait) en file d'attente est interrompue. IP vérifie le total de contrôle de l'en-tête IP pour s'assurer qu'il n'est pas altéré, et détermine si le paquet est destiné à ce système. Dans l'affirmative, et si la trame n'est pas un fragment, IP passe la chaîne mbuf à la routine d'entrée TCP ou UDP.

Si la trame reçue est un fragment de datagramme (ce qui ne peut se produire que dans UDP), IP bloque la trame. A réception des fragments suivants, ils sont fusionnés en un datagramme logique, lequel, une fois complet, est transmis à UDP. IP bloque les fragments d'un datagramme incomplet jusqu'à expiration du délai **ipfragttl** (spécifié via **no**). Le délai **ipfragttl** par défaut est de 30 secondes (valeur **ipfragttl** de 60). En cas de perte de fragments (suite à un incident réseau, une insuffisance d'espace mbuf, une saturation de la file de transmission, etc.), IP ne les reçoit jamais. A l'expiration de **ipfragttl**, IP rejette les fragments reçus. Ce qui est consigné par **netstat –s** sous l'intitulé "`ip:`" "`fragments dropped after timeout`"."

Couche IF (couche Demux sous AIX version 4.1)

Flux d'envoi

Lorsque la couche IF reçoit un paquet émis par IP, elle associe un en-tête de liaison de couche au début du paquet, vérifie que le format des mbuf est conforme aux spécifications d'entrée du pilote d'unité, puis appelle la routine d'écriture du pilote de l'unité.

Flux de réception

Sous AIX version 3, lorsque la couche IF reçoit un paquet du pilote d'unité, elle supprime l'en-tête de liaison, place la chaîne mbuf sur la file d'attente d'entrée IP (par le biais de pointeurs, et non par copie) et planifie une interruption hors niveau pour exécuter le traitement d'entrée IP.

Sous AIX version 4.1, lorsque la couche demux reçoit un paquet du pilote d'unité, elle appelle IP sur la routine d'interruption pour exécuter le traitement d'entrée IP.

Cartes réseau local et pilotes d'unité

Différents types de cartes réseau sont acceptés dans l'environnement AIX. Ces cartes se distinguent non seulement par les protocoles et supports de communication qu'elles prennent en charge, mais également par leur interface avec le bus d'E/S et le processeur. De même, les pilotes d'unité se différencient par la technique de transfert des données entre la mémoire et les cartes. La description ci-après s'applique globalement à la plupart des cartes et des pilotes d'unité, quelques points de détail étant néanmoins différents.

Flux d'envoi

Au niveau de la couche du pilote d'unité, la chaîne mbuf contenant le paquet est placée sur la file de transmission. Le nombre maximal de tampons de sorties qui peut être en file d'attente est contrôlé par le paramètre système **xmt_que_size**. Dans certains cas, les données sont copiées dans les tampons DMA du pilote. La carte est ensuite appelée à démarrer les opérations DMA.

A ce niveau, le contrôle revient à la routine de sortie TCP ou UDP, laquelle continue à émettre tant qu'elle a des données à envoyer. Le contrôle revient ensuite à l'application, qui s'exécute en mode asynchrone pendant que la carte transmet des données. Une fois la transmission terminée, la carte interrompt le système et les routines d'interruption d'unité sont appelées pour adapter les files de transmission et libérer les mbuf qui contenaient les données transmises.

Flux de réception

Lorsque des trames sont reçues par une carte, elles sont transférées dans une file de réception gérée par le pilote. Cette file d'attente peut être constituée de mbuf ou d'un pool de tampons distinct géré par le pilote d'unité. Dans les deux cas, les données se trouvent dans une chaîne mbuf lorsqu'elles passent du pilote d'unité à la couche IF.

Certains pilotes reçoivent les trames DMA dans une zone de mémoire fixe. Ils affectent ensuite des mbuf et y copient les données. Les pilotes/cartes recevant des trames MTU volumineuses peuvent réceptionner les trames DMA directement dans des grappes mbuf. Le pilote passe le contrôle de la trame au protocole approprié (IP dans cet exemple) en appelant une fonction de démultiplexage qui identifie le type de paquet et place le mbuf contenant le tampon dans la file d'entrée de ce protocole. A défaut de mbuf disponible, ou si la file d'entrée du plus haut niveau est saturée, les trames entrantes sont rejetées.

Optimisation de TCP et UDP

Les valeurs optimales des paramètres de communication varient selon le type de réseau et les caractéristiques des E/S de communication du système dominant et des programmes d'application. Les sections ci–après décrivent les principes généraux d'optimisation des communications, et poursuivent avec des conseils spécifiques destinés à différents types de réseaux locaux.

Recommandations

Vous pouvez viser soit un débit maximal, soit une utilisation minimale de la mémoire. Les conseils qui suivent s'appliquent à l'une ou l'autre des approches, parfois aux deux.

Optimisation du débit de sortie

Protocoles requête-réponse

- Pour maximiser le nombre de transactions par seconde, optez pour des messages courts.
- Pour maximiser le nombre d'octets par seconde, optez pour des messages supérieurs ou égaux à 1000 octets et juste inférieurs ou égaux à un multiple de 4096 octets.
- Si les requêtes et les réponses sont de taille fixe et tiennent dans un seul datagramme, passez par UDP :
  - Si possible, définissez des tailles d'écriture qui soient des multiples de la taille du MTU diminuée de 28 octets pour laisser la place aux en-têtes standard IP et UDP.
  - Il est plus efficace qu'une application écrive des messages longs, qui sont ensuite fragmentés et réassemblés par IP, plutôt que de multiplier les écritures.
  - Chaque fois que possible, faites appel à la routine **connect** pour associer une adresse au socket UDP. Cette opération peut être impossible sur un serveur communiquant avec plusieurs clients via un seul socket.
- Si les requêtes et les réponses sont de taille variable, passez par TCP assorti de l'option TCP_NODELAY. Nous avons mesuré que la charge de TCP est négligeable par rapport à celle d'UDP, notamment si des tailles d'écriture optimales sont spécifiées.
  - Pour éviter que des données ne soient copiées dans le noyau, définissez des tailles d'écriture supérieures à 936 octets.
  - Optez pour des écritures de taille égale ou légèrement inférieures à un multiple de la taille MTU. Ceci pour éviter l'envoi d'un segment (paquet) ne contenant que quelques octets.

Flux

- TCP offre un débit de sortie supérieur à celui d'UDP et assure la fiabilité de la transmission.
- La taille des écritures doit être un multiple de 4 096 octets. Si possible, les écritures doivent être de la taille du MSS (voir "Optimisation de la taille maximum de segment (MSS) TCP").

Optimisation de la mémoire

- Si votre trafic est essentiellement local, optez pour la plus grande taille de MTU acceptée par le type de votre réseau : vous minimiserez la fragmentation de paquets échangés par les systèmes locaux. Avec, en contrepartie, une fragmentation dans les passerelles connectant votre réseau à d'autres réseaux locaux dotés de MTU de taille inférieure (voir "Optimisation de la taille maximum de segment (MSS) TCP").
- Chaque fois que possible, les programmes d'application doivent lire et écrire des volumes :

– soit inférieurs ou égaux à 935 octets,

– soit égaux ou légèrement inférieurs à 4 096 octets (ou des multiples) :

Le premier sera placé dans les mbufs 1 à 4, les suivants feront bon usage des grappes de 4 096 octets utilisées pour les écritures de taille supérieure à 935 octets. Ecrire 936 octets entraîne un gaspillage de 3 160 octets par écriture : l'application peut saturer **udp_recvspace** (avec sa valeur par défaut de 65 536) avec 16 écritures – ne totalisant que 14 976 octets de données.

Passer par TCP gaspille temps et mémoire. TCP tente de former des données délimitées en paquets de la taille du MTU. Si le MTU du réseau est plus grand que 14 976 octets, TCP place la routine d'envoi à l'état veille dès que la limite de **tcp_sendspace** est atteinte. Il reçoit un acquittement (ACK) de dépassement de délai du récepteur pour forcer l'écriture des données.

Remarque : Si vous modifiez des paramètres via la commande **no**, les modifications ne sont effectives que jusqu'à l'amorçage système suivant : les paramètres sont alors réinitialisés à leur valeur par défaut. Pour pérenniser des modifications, placez la commande **no** appropriée dans le fichier /**etc**/**rc.net**.

Optimisation des files de transmission et de réception de la carte

La plupart des gestionnaires de communication proposent une série de paramètres optimisables qui permettent de contrôler les ressources de transmission et de réception. Ces paramètres déterminent en général les limites des files d'attente de transmission et de réception, mais ils peuvent également contrôler le nombre et la taille des tampons ou d'autres ressources. La limitation porte sur le nombre de tampons ou de paquets qui peuvent être placés en file d'attente avant transmission, ou sur le nombre de tampons disponibles pour la réception de paquets. Ces paramètres peuvent être optimisés de manière à permettre un placement suffisant en file d'attente au niveau de la carte, afin de gérer les pointes de charge générées par le système ou le réseau.

Files d'attente de transmission

Les gestionnaires de périphériques peuvent limiter la "file de transmission". Les limites peuvent être matérielles ou logicielles, selon le type de gestionnaire et de carte. Certains gestionnaires ne proposent qu'une file d'attente matérielle, alors que d'autres offrent des files matérielles et logicielles. D'aucuns autorisent un contrôle de la file matérielle en interne et la modification des limites de la file logicielle. En principe, le gestionnaire de périphérique place le paquet à transmettre directement dans la file d'attente matérielle de la carte. Si le CPU système est rapide par rapport au réseau, ou sur un système SMP, il peut arriver que le système produise les paquets à transmettre plus rapidement que le réseau ne peut les absorber. La file d'attente matérielle sera donc saturée. Dans ce cas de figure, certains gestionnaires proposent alors une file d'attente logicielle qui accueille les paquets en attente de transmission. Lorsque la limite de la file logicielle est atteinte, les paquets à transmettre sont annulés. Les performances peuvent s'en trouver affectées puisque les protocoles de haut niveau doivent retransmettre le paquet.

Avant AIX version 4.2.1, les limites supérieures des files de transmission étaient comprises entre 150 et 250, selon les cartes. Les valeurs système par défaut étaient généralement faibles, de l'ordre de 30. Depuis AIX version 4.2.1, les limites des files de transmission ont été portées sur la plupart des gestionnaires à 2048 tampons. Les valeurs par défaut ont également suivi cette augmentation, jusqu'à 512 pour la plupart des gestionnaires. Ces valeurs par défaut ont été augmentées car les systèmes SMP et CPU ayant gagné en rapidité, ils arrivaient à saturation avec les limites plus faibles autrefois définies.

Dans le cas de cartes présentant des limites de files matérielles, la modification de ces valeurs entraîne une augmentation de la consommation de mémoire réelle en raison des blocs de contrôle et tampons qui leur sont associés. Ces limites ne doivent donc être rehaussées que si le besoin s'en fait sentir ou, sur les gros systèmes, si l'accroissement de l'utilisation de la mémoire est négligeable. Dans le cas de limites de files de transmission logicielles, l'augmentation de ces valeurs ne joue pas sur l'utilisation de la mémoire. Elle permet seulement de mettre en file d'attente des paquets déjà alloués par les protocoles des couches hautes.

Files d'attente de réception

Certaines cartes vous permettent de configurer le nombre de ressources à utiliser pour la réception des paquets en provenance du réseau. Ces ressources peuvent définir le nombre de tampons de réception (et leur taille) ou consister simplement en un paramètre de file de réception (qui contrôle indirectement le nombre des tampons de réception).

Il peut s'avérer nécessaire d'augmenter les ressource de réception afin de gérer les pointes de charge du réseau.

AIX 3.2.x et AIX 4.x

Avec les versions AIX 3.2.x, les gestionnaires permettaient à des applications spéciales de lire les paquets reçus directement depuis le gestionnaire de périphérique. Le gestionnaire de périphérique gérait une 'file d'attente de réception' où ces paquets étaient placés en attente. Un paramètre de taille de la file de réception permettait alors de limiter le nombre de paquets pouvant être placés en attente pour ces applications. Sous AIX version 4.1, cette interface n'est plus prise en charge, sauf pour les anciennes cartes MicroChannel à condition que bos.compat LPP soit installé. Pour plus d'informations sur des cartes spécifiques, reportez–vous à la table des paramètres.

Pour les cartes PCI et les cartes Microchannel plus récentes, les paramètres de file de réception déterminent le nombre de tampons de réception mis à la disposition de la carte pour la réception des paquets entrants.

Tampons spécifiques au périphérique

AIX version 4.1.4 et ultérieures prennent en charge des Mbufs spécifiques au périphérique. Ce dernier peut ainsi affecter son propre jeu de tampons et le pré–configurer à l'intention de DMA. Les performances s'en trouvent améliorées puisque la charge supplémentaire induite par le mappage DMA n'intervient qu'une seule fois. Par ailleurs, la carte peut affecter les tailles de tampon les plus adaptées à sa taille de MTU. Ainsi, ATM, HIPPI et le commutateur SP2 acceptent une taille de MTU (paquet) de 64 ko. La taille maximale de mbuf système est de 16 Ko. En accordant à la carte des tampons de 64 ko, vous permettez l'écriture directe de grands blocs de 64 ko dans les tampons de 64 ko détenus par la carte, au lieu de leur copie vers plusieurs tampons de 16 ko (ce qui permet une meilleure affectation de la charge et libère les tampons excédentaires).

Les cartes acceptant des Mbuf spécifiques aux périphériques sont MCA ATM, MCA HIPPI et les différentes cartes SP2 à grande vitesse.

Le revers de la médaille de ces tampons spécifiques au périphérique est un niveau de complexité accru pour l'administrateur système. L'administrateur doit faire appel à des commandes spécifiques au périphérique pour consulter les statistiques liées aux tampons de carte et modifier les paramètres en conséquence. Si les statistiques révèlent que certains paquets ont été annulés par manque de ressources en tampons, les tailles de tampons doivent être augmentées.

Compte tenu de la diversité des gestionnaires et des utilitaires servant à modifier leurs paramètres, ils ne sont pas décrits dans ce manuel. Les paramètres MCA ATM sont répertoriés dans le tableau ci–après. Les statistiques ATM peuvent être consultées à l'aide de la commande `atmstat -d atm0` (substituez votre numéro d'interface à celui de l'exemple).

Quand les paramètres doivent–ils être augmentés ?

- Lorsque le CPU est plus rapide que le réseau et que plusieurs applications utilisent le même réseau. Ce cas de figure est classique sur un grand système multiprocesseur (SMP).
- Lorsque les options de la commande ”no” définissent des valeurs élevées pour `tcp_sendspace` ou `tcp_recvspace`, ou lors de l’exécution d’applications pouvant émettre des appels système afin d’augmenter la taille des tampons de socket d’envoi et de réception TCP. Ces valeurs élevées peuvent entraîner la transmission par le CPU d’un grand nombre de paquets vers la carte, ces paquets devant alors être placés en attente. Les procédures sont identiques pour `udp_sendspace` et `udp_recvspace` pour les applications UDP.
- Lorsque le trafic est très intermittent (en rafales).
- Un trafic important de petits paquets peut consommer plus de ressources qu’un trafic important de grands tampons. Ceci est dû au fait que les grands tampons étant plus longs à envoyer sur le réseau, le débit des paquets est plus lent pour les gros paquets.

Comment savoir si les paramètres doivent être augmentés ?

Plusieurs utilitaires d’état peuvent servir à cet effet. A partir d’AIX version 4.1.0, les statistiques de la carte indiquent le niveau haut de la file de transmission et le nombre de dépassements. Vous pouvez utiliser `netstat -v`, ou faire appel directement aux utilitaires statistiques de la carte (entstat pour Ethernet, tokstat pour Token Ring, fddistat pour FDDI, atmstat pour ATM, etc.)

Voici par exemple le résultat de la commande `entstat -d en0`. Il indique les statistiques concernant `en0`. L’option **–d** génère toutes les statistiques détaillées pour cette carte et doit être utilisée pour garantir l’affichage de l’intégralité des statistiques. Le champ ”Max Packets on S/W Transmit Queue:” indique le niveau haut de la file de transmission. Le champ ”S/W Transmit Queue Overflow:” indique le nombre de dépassements de la file d’attente logicielle.

> **Remarque :** Ces valeurs peuvent faire référence à la ”file d’attente matérielle” si la carte ne propose pas de ”file d’attente logicielle”. S’il existe des dépassements de la file de transmission, il convient d’augmenter les limites de la file matérielle ou logicielle pour ce gestionnaire.

Des ressources insuffisantes en réception sont indiquées par le champ ”Packets Dropped:” et, selon le type de carte, sont signalées par ”Out of Rcv Buffers” ou ”No Resource Errors:” ou toute autre indication comparable.

```
ETHERNET STATISTICS (en1) :
Device Type: IBM 10/100 Mbps Ethernet PCI Adapter (23100020)
Hardware Address: 00:20:35:b5:02:4f
Elapsed Time: 0 days 0 hours 7 minutes 22 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 1869143                              Packets: 1299293
Bytes: 2309523868                             Bytes: 643101124
Interrupts: 0                                 Interrupts: 823040
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
                                              Bad Packets: 0
Max Packets on S/W Transmit Queue: 41
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 1

Broadcast Packets: 1                          Broadcast Packets: 0
Multicast Packets: 0                          Multicast Packets: 0
No Carrier Sense: 0                           CRC Errors: 0
DMA Underrun: 0                               DMA Overrun: 0
Lost CTS Errors: 0                            Alignment Errors: 0
Max Collision Errors: 0                       No Resource Errors: 0
Late Collision Errors: 0                      Receive Collision Errors: 0
Deferred: 3778                                Packet Too Short Errors: 0
SQE Test: 0                                   Packet Too Long Errors: 0
Timeout Errors: 0                             Packets Discarded by Adapter: 0
Single Collision Count: 96588                 Receiver Start Count: 0
Multiple Collision Count: 56661
Current HW Transmit Queue Length: 1

General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Driver Flags: Up Broadcast Running
         Simplex 64BitSupport
```

Une autre méthode consiste à utiliser l'utilitaire 'netstat –i'. Si cet utilitaire indique une valeur non nulle dans la colonne "Oerrs" pour une interface, cela signifie que vous êtes en présence d'un dépassement de file d'attente en sortie. Ceci est valable pour toutes les versions d'AIX.

Comment connaître la valeur des paramètres ?

Vous pouvez faire appel à la commande de liste des attributs `lsattr -E -l [nom-carte]` ou utiliser SMIT pour afficher la configuration de la carte.

Selon les cartes, les noms des variables diffèrent. Ainsi, le paramètre correspondant à la file de transmission peut être ”sw_txq_size”, ”tx_que_size” ou ”xmt_que_size” pour n’en citer que quelques–uns. De même, les paramètres correspondant au pool de tampons et à la taille de la file de réception peuvent être appelés ”rec_que_size”, rx_que_size”ou encore ”rv_buf4k_min”.

Voici le résultat de la commande `lsattr -E -l atm0` sur une carte PCI 155 Mbs ATM. Il indique que `sw_txq_size` a la valeur `250` et que les tampons de réception `rv_buf4K_min` ont la valeur `x30`.

```
==== lsattr -E ========
dma_mem        0x400000    N/A                                          False
regmem         0x1ff88000  Bus Memory address of Adapter Registers      False
virtmem        0x1ff90000  Bus Memory address of Adapter Virtual Memory False
busintr        3           Bus Interrupt Level                          False
intr_priority  3           Interrupt Priority                           False
use_alt_addr   no          Enable ALTERNATE ATM MAC address              True
alt_addr       0x0         ALTERNATE ATM MAC address (12 hex digits)     True
sw_txq_size    250         Software Transmit Queue size                  True
max_vc         1024        Maximum Number of VCs Needed                  True
min_vc         32          Minimum Guaranteed VCs Supported              True
rv_buf4k_min   0x30        Minimum 4K-byte pre-mapped receive buffers    True
interface_type 0           Sonet or SDH interface                        True
adapter_clock  1           Provide SONET Clock                           True
uni_vers       auto_detect N/A                                          True
```

Voici un exemple de paramètres pour Microchannel 10/100 Ethernet généré en utilisant `lsattr -E -l ent0`. Il indique que `tx_que_size et rx_que_size` ont tous deux la valeur `256`.

```
bus_intr_lvl  11              Bus interrupt level                 False
intr_priority 3               Interrupt priority                  False
dma_bus_mem   0x7a0000        Address of bus memory used for DMA  False
bus_io_addr   0x2000          Bus I/O address                     False
dma_lvl       7               DMA arbitration level               False
tx_que_size   256             TRANSMIT queue size                  True
rx_que_size   256             RECEIVE queue size                   True
use_alt_addr  no              Enable ALTERNATE ETHERNET address    True
alt_addr      0x              ALTERNATE ETHERNET address           True
media_speed   100_Full_Duplex Media Speed                          True
ip_gap        96              Inter-Packet Gap                     True
```

Comment modifier les paramètres ?

Le moyen le plus simple consiste à détacher l’interface (ifconfig en0 detach, par exemple) puis à utiliser SMIT –> devices –> communications –> [type carte] –> change/show... pour afficher les paramètres de la carte. Une fois les paramètres affichés, placez le curseur sur celui que vous souhaitez modifier et appuyez sur F4 pour voir la fourchette des valeurs minimale et maximale pour ce champ ou les différentes tailles possibles. Vous pouvez sélectionner l’une de ces tailles et appuyez sur Entrée pour valider la requête et mettre à jour la base de données ODM. Remettez la carte en ligne (ifconfig en0 [nomhôte], par exemple). L’autre méthode permettant de modifier les paramètres est la commande chdev.

```
chdev -l [ifname] -a [nom-attribut]=nouvelle_valeur
```

Par exemple, pour ramener la valeur du paramètre `tx_que_size` ci–dessus sur `en0` à 128, utilisez la séquence de commandes suivante. Vous noterez que ce gestionnaire n’acceptant que quatre tailles différentes, il est préférable de faire appel à SMIT pour voir quelles sont ces valeurs.

```
ifconfig en0 detach
chdev -l ent0 -a tx_que_size=128
ifconfig en0 [hostname] up
```

Les informations qui suivent documentent les différents paramètres d'optimisation de carte. Ces paramètres et les valeurs indiquées concernent AIX 4.3.1 et peuvent être soumis à modification. Ils sont destinés à vous aider à comprendre les divers paramètres, ou si le système n'est pas disponible, à vous permettre de voir ces paramètres.

Les noms des paramètres, leurs valeurs par défaut et les fourchettes de valeurs sont issus de la base de données ODM. Le champ de commentaires est issu de la commande `lsattr -E -l nom-interface`.

Le champ **Remarques** contient des commentaires supplémentaires.

Carte MicroChannel (MCA)

```
Feature Code:  2980     (codename can't say)
Ethernet High-Performance LAN Adapter (8ef5)

Parameter    Default  Range    Comment                      Notes
-----------  -------- --------  ------------------------- ----------
xmt_que_size   512   20-2048   TRANSMIT queue size      SW TX queue
rec_que_size   30    20-150    RECEIVE queue size       See Note 1
rec_pool_size  37    16-64     RECEIVE buffer pool size On Adapter
```

Remarques :

1. Il s'agit d'une file de réception logicielle fournie uniquement à des fins de compatibilité avec les applications AIX 3.2.x utilisant l'interface du gestionnaire de périphérique réseau pour lire les paquets directement à partir du gestionnaire. Cette file limite le nombre de paquets entrants qui peuvent être placés en attente avant d'être reçus par ces applications. Ce paramètre n'est défini que si **bos.compat** est installé.

   Cette file d'attente n'est pas utilisée par la pile TCP/IP normale.

```
Feature Code: 2992      (codename Durandoak)
Ethernet High-Performance LAN Adapter (8f95)

Parameter      Default   Range   Comment              Notes
-------------  --------- -------- -------------------  ----------
xmt_que_size   512      20-2048  TRANSMIT queue size  SW queue


Feature Code: 2994      (codename SanRemo)
IBM 10/100 Mbps Ethernet TX MCA Adapter (8f62)

Parameter   Default  Range             Comment               Notes
----------- -------- ----------------  -------------------- ----------
tx_que_size   64     16,32,64,128,256  TRANSMIT queue size    HW queue
rx_que_size   32     16,32,64,128,256  RECEIVE queue size     HW queue


Feature Code: 2970      (codename Monterey)
Token-Ring High-Performance Adapter (8fc8)

Parameter      Default  Range    Comment                  Notes
-------------  -------- --------  ---------------------  ------------
xmt_que_size  99       32-2048   TRANSMIT queue size     SW queue
rec_que_size  30       20-150    RECEIVE queue size      See Note 1
```

```
Feature Code: 2972      (codename Wildwood)
Token-Ring High-Performance Adapter (8fa2)

Parameter      Default  Range    Comment                          Notes
-------------- -------- -------- -------------------------------  ----------
xmt_que_size   512      32-2048  TRANSMIT queue size              SW queue
rx_que_size    32       32-160   HARDWARE RECEIVE queue size      HW queue


Feature Code: 2727      (codename Scarborgh)
FDDI Primary Card, Single Ring Fiber

Parameter      Default  Range    Comment                           Notes
-------------- -------- --------  ------------------------------ -----------
tx_que_size    512      3-2048    Transmit Queue Size (in mbufs)
rcv_que_size   30       20-150    Receive Queue                   See Note 1


Feature Code: 2984      (codename Bumelia)
100 Mbps ATM Fiber Adapter (8f7f)

Parameter      Default   Range       Comment                      Notes
-------------- -------  ---------- ------------------------------ -----
sw_queue         512     0-2048  Software transmit queue len. SW Queue
dma_bus_width   0x1000000 0x800000-0x40000000,0x100000
                                 Amount of memory  to map for DMA See Note 3
max_sml_bufs    50      40-400    Maximum Small ATM mbufs   Max 256 byte buffers
max_med_bufs    100     40-1000   Maximum Medium ATM mbufs  Max 4KB buffers
max_lrg_bufs    300     75-1000   Maximum Large ATM mbufs   Max 8KB buffers,
                                                              See note 2
max_hug_bufs    50      0-400     Maximum Huge ATM mbufs    Max 16KB buffers
max_spec_bufs   4       0-400     Maximum ATM MTB mbufs    Max of max_spec_bufsize
spec_buf_size   64      32-1024   Max Transmit Block (MTB)  size kbytes)
sml_highwater   20      10-200    Minimum Small ATM mbufs   Min 256 byte buffers
med_highwater   30      20-300    Minimum Medium ATM mbufs  Min 4KB buffers
lrg_highwater   70      65-400    Minimum Large ATM mbufs   Min 8KB buffers
hug_highwater   10      4-300     Minimum Huge ATM mbufs    Min 16KB buffers
spec_highwater  20      0-300     Minimum ATM MTB mbufs     Min 64KB buffers
best_peak_rate  1500    1-155000  Virtual Circuit Peak Segmentation Rate
```

2. MCA ATM, le côté **rcv** (réception) utilise également de grands tampons (8 ko). La logique de réception n'utilise que des tampons de 8 ko ; par conséquent, si cette taille se révèle trop faible, les performances seront affectées.

   Les autres tailles de tampons ne sont destinées qu'aux tampons de transmission.

3. MCA ATM, Si vous augmentez le nombre total de tampons, vous aurez peut–être à modifier le paramètre `dma_bus_width = 0x1000000`. La largeur du bus mémoire DMA détermine la quantité totale de mémoire utilisée pour les tampons ATM. Augmentez la valeur de ce paramètre si vous recevez un message d'erreur lorsque vous tentez d'augmenter le nombre maximum de tampons ou les limites du niveau haut.

```
Feature Code: 2989        (codename Clawhammer)
155 Mbps ATM Fiber Adapter (8f67)

Parameter      Default   Range     Comment    Notes
-------------- --------  --------  ---------- -------
 (same as ATM 100 adapter above)
```

Cartes PCI

```
Feature Code 2985        (codename Klickitat)
IBM PCI Ethernet Adapter (22100020)

Parameter    Default  Range            Comment              Notes
------------ -------- ---------------- ------------------- ---------
tx_que_size   64      16,32,64,128,256  TRANSMIT queue size HW Queues
rx_que_size   32      16,32,64,128,256  RECEIVE queue size  HW Queues


Feature Code 2968         (codename Phoenix)
IBM 10/100 Mbps Ethernet PCI Adapter (23100020)

Parameter      Default   Range            Comment                Notes
-------------- ------- ---------------- ------------------- ------------
tx_que_size     256    16,32,64,128,256 TRANSMIT queue size   HW Queue Note 1
rx_que_size     256    16,32,64,128,256 RECEIVE queue size    HW Queue Note 2
rxbuf_pool_size 384    16-2048     # buffers in receive  Dedicated receive
                                     buffer pool           buffers Note 3
```

Remarques sur Phoenix :

1. Avant AIX 4.3.2, la valeur par défaut de tx_queue_size était 64
2. Avant AIX 4.3.2, la valeur par défaut de rx_queue_size était 32
3. AIX 4.3.2, le gestionnaire a ajouté un nouveau paramètre pour contrôler le nombre de tampons dédiés à la réception de paquets.

```
Feature Code: 2969       (codename Galaxy)
Gigabit Ethernet-SX PCI Adapter (14100401)

Parameter     Default Range    Comment                                Notes
------------- ------- -------- ---------------------------------- ---------
tx_que_size   512  512-2048    Software Transmit Queueu size      SW Queue
rx_que_size   512     512      Receive queue size                 HW Queue
receive_proc  6       0-128    Minimum Receive Buffer descriptors


Feature Code: 2986       (codename Candlestick)
3Com 3C905-TX-IBM Fast EtherLink XL NIC

Parameter      Default  Range  Comment                          Notes
-------------- -------- ------ ------------------------------   ----------
tx_wait_q_size   32     4-128  Driver TX Waiting Queue Size   HW Queues
rx_wait_q_size   32     4-128  Driver RX Waiting Queue Size   HW Queues


Feature Code: 2742       (codename Honeycomb)
SysKonnect PCI FDDI Adapter (48110040)

Parameter     Default  Range    Comment               Notes
------------- -------- -------- ------------------- ---------------
tx_queue_size   30     3-250    Transmit Queue Size   SW Queue
RX_buffer_cnt   42     1-128    Receive frame count Rcv buffer pool


Feature Code: 2979       (codename Skyline)
IBM PCI Tokenring Adapter (14101800)

Parameter     Default  Range    Comment               Notes
------------- -------- ------- --------------------------- --------
xmt_que_size  96       32-2048 TRANSMIT queue size         SW Queue
rx_que_size   32       32-160  HARDWARE RECEIVE queue size HW queue
```

```
Feature Code: 2979       (codename Cricketstick)
IBM PCI Tokenring Adapter (14103e00)

Parameter     Default  Range    Comment               Notes
------------- -------- -------- -------------------- --------
xmt_que_size  512      32-2048  TRANSMIT queue size  SW Queue
rx_que_size   64       32-512   RECEIVE queue size   HW Queue


Feature Code: 2988       (codename Lumbee)
IBM PCI 155 Mbps ATM Adapter (14107c00)

Parameter     Default   Range      Comment                           Notes
------------- --------- ---------- --------------------------------- -------
sw_txq_size   100       0-4096     Software Transmit Queue size    SW Queue
rv_buf4k_min  48(0x30) 0-512(x200) Minimum 4K-byte pre-mapped receive Buffers
```

Optimisation de la taille maximum de segment (MSS) TCP

Le protocole TCP comporte un mécanisme permettant aux deux extrémités d'une connexion de négocier la taille maximale de segment (MSS) à utiliser sur la liaison. Chaque extrémité se sert du champ OPTIONS de l'en–tête TCP pour proposer un MSS. Le MSS choisi est la plus petite des valeurs proposées par les deux extrémités.

Le but de cette négociation est d'éviter les délais d'attente et la baisse de débit consécutives à la fragmentation des paquets lorsqu'ils passent par des routeurs ou des passerelles pour n'être réassemblés que sur l'hôte destinataire.

La valeur de MSS indiquée par le logiciel TCP au cours de la configuration de la connexion diffère selon que l'autre extrémité est un système *local,* sur le même réseau physique (c'est-à-dire ayant le même numéro de réseau), ou un système d'un réseau *distant.*

Réseau local

Si l'autre système est local, le MSS notifié par TCP est basé sur le MTU (maximum transfer unit) de l'interface du réseau local :

```
TCP MSS = MTU - taille en-tête TCP - taille en-tête IP.
```

Il s'agit du maximum possible sans fragmentation IP : cette valeur est donc optimale par définition, et aucune optimisation du MSS n'est requise pour les réseaux locaux.

Réseau distant

Lorsque l'autre extrémité se trouve sur un réseau distant, TCP sous AIX spécifie pour MSS une valeur par défaut de 512 octets. Cette valeur conservatoire est basée sur le fait que tous les routeurs IP prennent en charge un MTU d'au moins 576 octets.

Le MSS optimal pour les réseaux distants est basé sur le plus petit MTU des réseaux intervenant sur la route entre la source et la destination. Il s'agit généralement d'un nombre dynamique qui ne peut être établi que par une certaine recherche au niveau du MTU du chemin. Le protocole TCP ne fournit aucun mécanisme permettant de déterminer ce nombre, c'est pourquoi la valeur par défaut est couramment adoptée. Il est toutefois possible de lancer une recherche du PMTU TCP à l'aide de la commande :

```
no -o tcp_pmtu_discover=1"
```

L'un des effets secondaires de ce paramètre est l'augmentation de la table de routage (une entrée de plus par connexion TCP active). L'option de la commande no "route_expire" doit avoir une valeur non nulle afin que toute entrée de routage en mémoire cache non utilisée soit supprimée de la table à l'issue de la durée d'inactivité "route_expire".

Cette valeur par défaut, qui convient bien à Internet en général, peut se révéler inutilement restrictive pour les interréseaux d'un domaine administratif. Dans un environnement de ce type, les tailles des MTU des réseaux physiques constitutifs sont connues et le MTU minimal de même que le MSS optimal peuvent être déterminés par l'administrateur. AIX offre plusieurs moyens pour forcer TCP à utiliser ce MSS optimal. Les hôtes source et cible doivent tous deux prendre en charge ces fonctions. Dans un environnement hétérogène, multifournisseur, la disponibilité de la fonction sur les deux systèmes peut être un facteur déterminant dans le choix de la solution.

Routes statiques

Pour passer outre la valeur par défaut de MSS (512), il suffit de spécifier une route statique vers un réseau distant spécifique et d'associer l'option **–mtu** à la commande **route** pour spécifier le MTU à ce réseau. Dans ce cas, spécifiez le MTU minimal effectif de la route, au lieu de calculer une valeur MSS.

Dans un environnement stable et limité, cette méthode permet un contrôle précis du MSS réseau par réseau. Elle présente néanmoins quelques inconvénients :

- Elle est incompatible avec le routage dynamique.
- Elle devient peu pratique si le nombre de réseaux distants devient conséquent.
- Les routes statiques doivent être définies aux deux extrémités, pour que les deux côtés négocient avec un MSS supérieur au MSS par défaut.

Option tcp_mssdflt de la commande no

La valeur par défaut (512) adoptée par TCP pour les réseaux distants peut être modifiée via le paramètre **tcp_mssdflt** de la commande **no**. La modification s'applique à l'ensemble du système.

La valeur destinée à remplacer la valeur de MSS par défaut doit être la valeur minimale de MTU diminuée de 40 pour laisser la place aux en-têtes TCP et IP.

Dans un environnement où le MTU est supérieur au MTU par défaut, cette méthode offre l'avantage d'éliminer la nécessité de définir le MSS réseau par réseau. Inconvénients :

- Augmenter la valeur par défaut peut entraîner une fragmentation au niveau du routeur IP si la destination se trouve sur un réseau réellement éloigné et que les MTU des réseaux intervenants sont inconnus.
- Le paramètre **tcp_mssdflt** doit avoir la même valeur sur l'hôte destination.

Sous-réseau et option subnetsarelocal de la commande no

Il est possible de faire partager à plusieurs réseaux physiques le même numéro de réseau, par le biais de sous-réseaux (voir "Adressage TCP/IP"). L'option **subnetsarelocal** de la commande **no** spécifie, sur l'ensemble du système, si les sous-réseaux doivent être considérés comme locaux ou distants. Avec **subnetsarelocal=1** (valeur par défaut), l'hôte A sur le sous-réseau 1 considère l'hôte B sur le sous-réseau 2 comme se trouvant sur le même réseau physique.

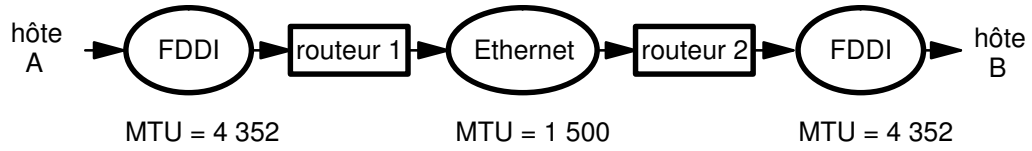
La conséquence est que lorsque les hôtes A et B établissent une connexion, ils négocient le MSS en supposant qu'ils se trouvent sur le même réseau. Chaque hôte indique un MSS sur la base du MTU de son interface réseau. Ce qui conduit généralement au choix d'un MSS optimal.

Cette approche présente plusieurs avantages :

- Elle ne requiert aucun liaison statique ; MSS est automatiquement négocié.
- Elle ne désactive ni ne supplante la négociation du MSS par TCP : de légères différences de MTU entre réseaux adjacents peuvent ainsi être correctement gérées.

Inconvénients :

- Risque de fragmentation au niveau du routeur IP lorsque deux réseaux à fort MTU sont reliés via un réseau à faible MTU. La figure ”Fragmentation inter–sous–réseaux” illustre ce problème.



Fragmentation inter-sous-réseaux

Dans ce scénario, les hôtes A et B établissement une connexion sur la base d’un MTU commun de 4 352. Un paquet transitant de A vers B sera fragmenté par le routeur 1 et défragmenté par le routeur 2 (et vice versa).

- Source et destination doivent considérer les sous-réseaux comme locaux.

Optimisation des performances du protocole IP

Au niveau de la couche IP, le seul paramètre optimisable est **ipqmaxlen,** qui contrôle la longueur de la file d’entrée IP – citée précédemment (qui n’existe que sous AIX version 3). Les paquets peuvent arriver très vite et saturer cette file d’entrée. Sous AIX, il n’y a aucun moyen simple de savoir si cela s’est produit. Seul un compteur de dépassement peut être affiché via la commande **crash**. Pour vérifier cette valeur, lancez la commande **crash** et, à l’invite, entrez : `knlist ipintrq`. Le résultat est une valeur hexadécimale, variable selon le système. Ajoutez ensuite `10` (hexa) à cette valeur et indiquez le total comme argument de la commande **od**. Par exemple :

```
# crash
> knlist ipintrq
  ipintrq: 0x0149ba68
> od 0149ba78 1
0149ba78: 00000000 <--    Valeur du compteur de déplacement de la
                          file d’entrée IP
>quit
```

Si le nombre résultant est strictement positif, un dépassement a eu lieu. La longueur maximale de cette file d’attente est définie via la commande **no**. Par exemple :

```
no –o ipqmaxlen=100
```

permet la mise en file d’attente de 100 paquets. La valeur exacte à utiliser est déterminée par la vitesse maximale de réception en rafale. Si celle-ci ne peut être déterminée, le nombre de dépassement peut aider à déterminer l’augmentation requise. L’augmentation de la taille de la file d’attente n’induit aucune consommation mémoire supplémentaire. Cette augmentation peut néanmoins allonger le temps passé dans le gestionnaire d’interruption hors niveau, dans la mesure où IP doit traiter davantage de paquets dans sa file d’entrée. Ce qui risque d’affecter le temps CPU requis par les processus. Le choix est entre un faible taux de rejet de paquets et la disponiblité de la CPU pour d’autres traitements. Le mieux est d’augmenter **ipqmaxlen** par petits incréments si le choix n’est pas évident.

Optimisation des performances Ethernet

Ethernet a contribué à l’élaboration de l’algorithme du ”plus petit commun dénominateur” pour le choix du MTU. Si une configuration inclut des réseaux Ethernet et d’autres réseaux locaux, et que le trafic est soutenu, les MTU de tous les réseaux devront peut-être être définis à 1 500 octets pour éviter la fragmentation lorsque les données arrivent sur un réseau Ethernet.

- La valeur par défaut (1 500), qui est aussi la valeur maximale de MTU, ne doit pas être modifiée.
- La taille de bloc de l'application doit être un multiple de 4 096 octets.
- Les paramètres relatifs à l'espace socket peuvent conserver leur valeur par défaut.
- Si la charge de travail prévoit une utilisation intensive de services faisant appel à UDP, tels que NFS ou RPC, il convient d'augmenter la valeur de **sb_max** pour prendre en compte le fait que chaque MTU de 1 500 octets utilise un tampon de 4 096 octets.

Optimisation des performances anneau à jeton (4 Mo)

La valeur par défaut de MTU (1 492 octets) convient aux réseaux en anneau à jeton interconnectés à des réseaux Ethernet ou à des réseaux hétérogènes dont le MTU minimal est inconnu.

- Sauf en cas de trafic intense entre le réseau et les réseaux externes, le MTU doit être augmenté à sa valeur maximale (3 900 octets).
- La taille de bloc de l'application doit être un multiple de 4 096 octets.
- Les paramètres relatifs à l'espace socket peuvent conserver leur valeur par défaut.
- Si la charge de travail prévoit une utilisation intensive de services faisant appel à UDP, tels que NFS ou RPC, il convient d'augmenter la valeur de **sb_max** pour prendre en compte le fait que chaque MTU de 1 492 octets utilise un tampon de 4 096 octets.

Optimisation des performances anneau à jeton (16 Mo)

La valeur par défaut de MTU (1 492 octets) convient aux réseaux en anneau à jeton interconnectés à des réseaux Ethernet ou à des réseaux hétérogènes dont le MTU minimal est inconnu.

- Sauf en cas de trafic intense entre le réseau et les réseaux externes, le MTU doit être augmenté à la valeur de 8 500 octets. Ce qui permet aux paquets NFS de 8 ko de tenir dans un seul MTU. Augmenter encore le MTU à son maximum de 17 000 octets entraîne rarement une amélioration significative du débit.
- La taille de bloc de l'application doit être un multiple de 4 096 octets.
- Les paramètres relatifs à l'espace socket peuvent conserver leur valeur par défaut.
- Si la charge de travail prévoit une utilisation intensive de services faisant appel à UDP, tels que NFS ou RPC, et que le MTU doit conserver sa valeur par défaut à cause des interconnexions, il convient d'augmenter la valeur de **sb_max** pour prendre en compte le fait que chaque MTU de 1 492 octets utilise un tampon de 4 096 octets.

Optimisation des performances FDDI

En dépit d'un MTU relativement faible, ce support haute vitesse tire parti des augmentations substantielles de la taille du tampon de socket.

- Sauf en cas de trafic intense entre le réseau et les réseaux externes, le MTU doit conserver sa valeur par défaut (4 352 octets).
- Lorsque possible, une application exploitant TCP doit écrire par tranches de multiples de 4 096 octets à la fois (8 ou 16 ko de préférence), pour un débit maximal.
- Spécifiez **no –o sb_max=***(2*nouvelle-taille)* pour relever le plafond de l'espace tampon du socket.
- Spécifiez **no –o *_*space=***nouvelle-taille* pour donner à l'espace d'envoi et de réception du socket TCP et UDP la valeur par défaut de *nouvel-espace*. Nouvel–espace doit être au moins égal à 57 344 octets (56 ko).

- Pour ESCALA modèle *90 ou plus rapide, utilisez **no –o rfc1323=1** pour que la taille des tampons de socket puisse être définie à une valeur supérieure à 64 ko. Suivez ensuite la procédure précédente avec *nouvelle-taille* au moins égal à 128 ko.

Optimisation des performances ATM

- Sauf en cas de trafic intense entre le réseau et les réseaux externes, le MTU doit conserver sa valeur par défaut (9 180 octets).
- Lorsque possible, une application exploitant TCP doit écrire par tranches de multiples de 4 096 octets à la fois (8 ou 16 ko de préférence), pour un débit maximal.
- Spécifiez **no –o sb_max=***(2*nouvelle-taille)* pour relever le plafond de l'espace tampon du socket.
- Spécifiez **no –o *_*space=***nouvelle-taille* pour donner à l'espace d'envoi et de réception du socket TCP et UDP la valeur par défaut de *nouvel-espace. nouvel–espace* doit être au moins égal à 57 344 octets (56 ko).
- Pour ESCALA modèle *90 ou plus rapide, utilisez **no –o rfc1323=1** pour que la taille des tampons de socket puisse être définie à une valeur supérieure à 64 ko. Suivez ensuite la procédure précédente avec *nouvelle-taille* au moins égal à 128 ko.

Optimisation des performances SOCC

- Le MTU doit conserver sa valeur par défaut (61 428 octets).
- Lorsque possible, une application exploitant TCP doit écrire par tranches de multiples de 28 672 octets à la fois (28 ko), pour un débit maximal.
- La valeur par défaut de l'espace d'envoi et de réception du socket TCP et UDP doit être définie à 57 344 octets.

Optimisation des performances HIPPI

- Le MTU doit conserver sa valeur par défaut (65 536 octets).
- Lorsque possible, une application exploitant TCP doit écrire par tranches de 65 536 octets à la fois, pour un débit maximal.
- Donnez à **sb_max** une valeur supérieure à 2*655 360.
- La valeur par défaut de l'espace d'envoi et de réception du socket TCP et UDP doit être définie à 655 360 octets. Spécifiez **no –o rfc1323=1** pour que la taille des tampons de socket puisse être définie à une valeur supérieure à 64 ko.

Optimisation du pool mbuf

Remarque : Cette section ne concerne que la version 3.2.5. Sous AIX version 4.1, le mécanisme d'affectation des mbuf est sensiblement différent. Vous pouvez fixer la quantité maximale de mémoire qui sera utilisée par le programme d'affectation du réseau, comme vous le faites sous la version 3.2.5, c'est-à-dire avec la commande **no** et le paramètre **thewall**. Les autres options d'optimisation disponibles sous la version 3.2.5 ont été supprimées de la version 4.1 car le mécanisme d'affectation des mbuf a une fonction d'optimisation automatique.

Le sous-système réseau utilise une fonction de gestion de la mémoire qui dépend d'une structure de données appelée *mbuf*. Les mbuf servent essentiellement à stocker des données entrantes et sortantes du réseau. Disposer de pools mbuf de la taille adéquate peut améliorer sensiblement les performances du réseau. A l'inverse, une configuration inadéquate des pools mbuf affecte les performances réseau et système. Le système d'exploitation AIX permet de configurer le pool mbuf en cours d'exploitation. Cet avantage suppose de savoir quand et comment adapter les pools.

C'est l'objet des sections suivantes :

- "Fonction de gestion des mbuf : généralités"
- "Optimisation des pools mbuf : quand ?"
- "Optimisation des pools mbuf : comment ?"

Fonction de gestion des mbuf : généralités

La fonction de gestion des mbuf contrôle deux pools de tampons : un pool de tampons de faible capacité (256 octets chacun), simplement appelés *mbuf*, et un pool de tampons de capacité élevée (4 096 octets chacun), généralement appelés *grappes mbuf* ou *grappes.* Les pools sont créés à partir de la mémoire système, par une requête d'affectation au gestionnaire de mémoire virtuelle (VMM). Les pools sont constitués d'éléments fixes de mémoire virtuelle – ce qui signifie qu'ils résident en permanence en mémoire physique et ne font jamais partie de pages évacuées. En conséquence, la capacité de mémoire réelle, disponible pour la pagination dans les programmes d'application et les données, est diminuée d'un volume équivalent à l'augmentation de la capacité des pools mbuf. Ne l'oubliez pas lorsque vous augmentez la taille des pools mbuf.

La taille initiale des pools mbuf dépend du système. Un nombre minimal de (petits) mbufs et de grappes est affecté à chaque système, mais ces minimums sont augmentés en fonction de la configuration spécifique des systèmes. Un des facteurs pris en compte est le nombre de cartes de communication équipant le système. La taille par défaut est initialement configurée pour gérer des réseaux de charge faible à moyenne (trafic de 100 à 500 paquets/seconde). La taille des pools augmente dynamiquement avec l'augmentation de la charge du réseau. Le pool de grappe diminue si la charge du réseau diminue (le pool mbuf n'est jamais réduit). Pour optimiser les performances du réseau, l'administrateur doit équilibrer la taille des pools mbuf en fonction de la charge du réseau (paquets/seconde). Si cette dernière est axée sur le trafic UDP (comme sur un serveur NFS, par exemple), la taille du pool mbuf doit être double du taux de paquets/seconde. Ceci parce que le trafic UDP consomme un petit mbuf supplémentaire.

Pour assurer l'efficacité du service d'affectation de mbuf, il est tentant de maintenir libres en permanence un minimum de tampons dans les pools. Les paramètres **lowmbuf** et **lowclust** permettent (via la commande **no**) de définir ces limites inférieures.

Le paramètre **lowmbuf** spécifie le nombre minimal de tampons libres du pool mbuf. Le paramètre **lowclust** spécifie le nombre minimal de tampons libres du pool de grappe. Lorsque le nombre de tampons passe en-deçà du seuil défini par **lowmbuf** ou **lowclust**, les pools sont agrandis. Cette extension n'est pas immédiate, mais planifiée pour être effectuée par un service du noyau appelé **netm**. Lorsque le service **netm** est distribué, les pools sont

agrandis de façon à répondre aux minimums définis par **lowclust** et **lowmbuf**. C'est la structure VMM qui impose que cette opération soit le fait d'un processus du noyau.

Le service de noyau **netm** limite en outre la croissance du pool de grappes. Le paramètre **mb_cl_hiwat** définit cette limite supérieure.

Le paramètre **mb_cl_hiwat** contrôle le nombre maximal de tampons libres que peut contenir le pool de grappes. Lorsque le seuil défini par **mb_cl_hiwat** est dépassé, **netm** est planifié pour libérer quelques grappes et les restituer à VMM.

Le système de noyau **netm** est exécuté selon une priorité élevée (fixée à 37). De ce fait, distribuer trop de **netm** peut avoir un effet négatif non seulement sur les performances du réseau, mais également sur celles du système – pour cause de conflit avec les autres processus utilisateur et système. Des pools mal configurés peuvent provoquer l'emballement des **netm**, dû à des incompatibilités entre les besoins du trafic réseau et des seuils mal définis. La distribution du système de noyau **netm** peut être minimisée en configurant les pools mbuf pour qu'ils répondent aux besoins du réseau et du système.

La fonction de gestion mbuf se sert encore d'un dernier paramètre réseau, **thewall**.

thewall contrôle la taille de RAM maximale (en kilo-octets) susceptible d'être affectée par la fonction de gestion mbuf à partir de VMM. Ce paramètre sert à éviter tout déséquilibre des ressources VMM, qui aurait un fâcheux effet sur les performances système.

Optimisation des polls mbuf : quand ?

Quand et de quelle quantité optimiser les pools mbuf est directement lié à la charge du réseau à laquelle est soumise une machine donnée. Un serveur gérant plusieurs clients fait partie des machines dont il est bon d'optimiser les pools mbuf. Pour ce faire, il est essentiel que l'administrateur connaisse bien la charge du réseau pour un système donné.

Vous pouvez, via la commande **netstat**, connaître approximativement la charge du réseau en paquets/seconde. Par exemple :

```
netstat -I tr0 5
```

indique le trafic en entrée et en sortie, issu de la carte `tr0` et de toutes les cartes LAN du système. Le listing ci-dessous indique l'activité entraînée par une vaste opération provoquée par une commande **ftp** :

```
$ netstat -I tr0 2
   input    (tr0)     output           input   (Total)    output
 packets  errs  packets  errs colls   packets  errs  packets  errs colls
   20615   227     3345     0     0     20905   227     3635     0     0
      17     0        1     0     0        17     0        1     0     0
     174     0      320     0     0       174     0      320     0     0
     248     0      443     0     0       248     0      443     0     0
     210     0      404     0     0       210     0      404     0     0
     239     0      461     0     0       239     0      461     0     0
     253     1      454     0     0       253     1      454     0     0
     246     0      467     0     0       246     0      467     0     0
      99     1      145     0     0        99     1      145     0     0
      13     0        1     0     0        13     0        1     0     0
```

La commande **netstat** peut être assortie d'un indicateur, **–m**, qui donne des détails sur l'usage et la disponibilité des mbuf et des grappes :

```
253 mbufs in use:
        50 mbufs allocated to data
        1 mbufs allocated to packet headers
        76 mbufs allocated to socket structures
        100 mbufs allocated to protocol control blocks
        10 mbufs allocated to routing table entries
        14 mbufs allocated to socket names and addresses
        2 mbufs allocated to interface addresses
16/64 mapped pages in use
319 Kbytes allocated to network (39% in use)
0 requests for memory denied
0 requests for memory delayed
0 calls to protocol drain routines
```

La ligne `16/64 mapped pages in use` indique qu'il existe 64 grappes fixes, dont 16 sont actuellement utilisées.

Ce compte rendu peut être comparé aux paramètres système définis, via la commande **no –a**. Les lignes intéressantes à cet égard sont :

```
    lowclust = 29
     lowmbuf = 88
     thewall = 2048
mb_cl_hiwat = 58
```

Il est clair que sur le système test, les `319 Kbytes allocated to network` sont très en-deçà de la valeur de **thewall** (2 048 ko) et que les (64 – 16 = 48) grappes libres sont également en-deçà de la limite **mb_cl_hiwat** (58).

Le compteur `requests for memory denied`, tenu à jour par la fonction de gestion des mbuf, est incrémenté chaque fois qu'une requête d'affectation de mbuf n'aboutit pas. Normalement, la valeur de `requests for memory denied` est 0. Si une surcharge de trafic se produit sur un système, les pools mbuf configurés par défaut peuvent se révéler insuffisants : le compteur d'erreur est alors incrémenté chaque fois qu'une requête d'affectation de mbuf échoue. Le compteur atteint rapidement des chiffres de l'ordre des milliers, compte tenu du nombre de paquets arrivant dans un court laps de temps. Les statistiques `requests for memory denied` correspondent aux paquets abandonnés sur le réseau. Ces paquets supposent des retransmissions, c'est-à-dire une baisse des performances. Si la valeur de `requests for memory denied` est supérieure à zéro, il peut être utile d'optimiser les paramètres mbuf — voir Optimisation des pools mbuf – comment ?.

Le décompte `Kbytes allocated to the network`, tenu à jour par la fonction de gestion des mbuf, représente la quantité de mémoire système actuellement affectée aux deux pools mbuf. La limite supérieure de ce décompte, définie par **thewall**, permet d'empêcher la fonction de gestion mbuf de consommer trop de mémoire physique sur un système. La valeur par défaut de **thewall** limite cette fonction à 2 048 ko (comme indiqué sur le comte rendu généré par la commande **no** –a). Si la valeur de `Kbytes allocated to the network` est proche de **thewall**, il peut être utile d'optimiser les paramètres mbuf. Reportez-vous à "Optimisation des pools mbuf : comment ?".

Il est des cas où les indicateurs ci-dessus laissent supposer qu'il faut étendre les pools mbuf, alors qu'en fait, il y a un problème système auquel il convient d'abord de remédier. Par exemple :

- manque de mémoire mbuf
- données en attente non lues à partir d'un socket ou d'une autre structure d'attente interne.

Une perte de mémoire mbuf dénote une situation dans laquelle un code noyau ou une extension noyau a omis de libérer une ressource mbuf et détruit le pointeur vers l'emplacement mémoire correspondant, perdant à tout jamais l'adresse du mbuf. Si cette situation est récurrente, toutes les ressources mbuf finiront par être inaccessibles. Si les statistiques mbuf générées par **netstat** montrent un accroissement progressif de l'usage des mbuf, n'accusant jamais de baisse, ou une utilisation intensive des mbuf sur un système relativement peu chargé, il y a sans doute une perte de mémoire mbuf.
Les développeurs d'extensions noyau se doivent de toujours intégrer dans leurs tests un contrôle sur les pertes de mémoire.

Il se peut également que de nombreux mbuf restent en attente dans un socket suite à une erreur dans une application. Normalement, un programme d'application lit les données d'un socket, entraînant la réattribution des mbuf à la fonction de gestion du mbuf.
Un administrateur peut gérer les statistiques générées par la commande **netstat –m** et chercher un éventuel usage intensif du mbuf alors que le trafic réseau ne le justifie pas. Il peut également examiner la liste des processus en cours (via `ps -ef`) et étudier de près ceux qui utilisent le sous-système réseau avec une forte consommation de temps CPU.
Il convient alors d'isoler l'application en cause et de la corriger.

Optimisation des polls mbuf : comment ?

Avec un minimum de connaissance sur leur organisation et leur gestion, optimiser les pools mbuf est, sous AIX, une opération simple, qui peut être effectuée en cours d'exploitation.
L'utilisateur racine dispose de la commande **no** pour modifier les paramètres du pool mbuf.
Voici quelques règles :

- Si vous modifiez les attributs **lowclust** et **lowmbuf**, vous devrez peut-être commencer par augmenter **thewall**, pour éviter que l'extension du pool n'atteigne le seuil **thewall**.
- Dans tous les cas, la valeur de l'attribut **mb_cl_hiwat** doit être au moins double de celle de **lowclust**. Ceci pour éviter tout emballement de **netm** (voir plus haut).
- Si vous modifiez **lowclust**, vous devez modifier **lowmbuf** d'une valeur au moins équivalente. Il existe un mbuf pointant sur chaque grappe.
- Après extension des pools, lancez la commande **vmstat** pour vérifier que les taux de pagination n'ont pas augmenté. Si vous ne parvenez pas à amener les pools au niveau souhaité sans pour autant affecter les taux de pagination, un supplément de mémoire est sans doute à prévoir.

Voici un exemple de script shell qui peut être exécuté à la fin de **/etc/rc.net** pour optimiser les pools mbuf pour un serveur NFS prenant en charge un trafic réseau d'environ 1500 paquets/seconde.

```
#!/bin/ksh
# echo ”Tuning mbuf pools...”
# set maximum amount of memory to allow for allocation (10MB)
no -o thewall=10240

# set minimum number of small mbufs
no -o lowmbuf=3000

# generate network traffic to force small mbuf pool expansion
  ping 127.0.0.1 1000 1 >/dev/null

# set minimum number of small mbufs back to default to prevent netm from
# running unnecessarily
no -d lowmbuf

# set maximum number of free clusters before expanding pool
# (about 6MB)
no -o mb_cl_hiwat=1500

# gradually expand cluster pool
N=10
while [ $N -lt 1500 ]
do
  no -o lowclust=$N
  ping 127.0.0.1 1000 1 >/dev/null
  let N=N+10
done

# set minimum number of clusters back to default to prevent netm
# from running unnecessarily
no -d lowclust
```

Vous pouvez lancer `netstat -m` à la suite du script ci-dessus pour vérifier la taille du pool de grappes (appelées pages mappées par la commande **netstat**). Pour vérifier la taille du pool de mbuf, vous disposez de la commande **crash** qui permet d'examiner une structure de données noyau, **mbstat** (voir le fichier /**usr**/**include**/**sys**/**mbuf.h**). L'adresse du noyau de **mbstat** peut être affichée dans le cadre de **crash**, via la commande **od mbstat**. Entrez ensuite `od <adresse-noyau>` pour obtenir un cliché du premier mot de la structure **mbstat**, qui contient la taille du pool mbuf. Si vous utilisez AIX version 4.1 ou version 3.2 avec pTF U437500, le dialogue est semblable à :

```
$ crash
> od mbstat
001f7008: 00000180
> quit
```

La taille du pool mbuf est ainsi de 18016 (38410).

Si vous utilisez AIX version 3.2 sans le PTF U437500, le dialogue est semblable à :

```
$ crash
> od mbstat
000e2be0: 001f7008
> od 1f7008
001f7008: 00000180
> quit
```

La taille du pool mbuf est ainsi de 18016 (38410).

Récapitulatif des paramètres d'optimisation UDP, TCP/IP et mbuf

Les paragraphes suivants répertorient tous les attributs et techniques d'optimisation pour chacun des paramètres d'optimisation des communications.

thewall

Objet : Limite supérieure absolue de la quantité de mémoire réelle susceptible d'être utilisée par le sous-système de communication.

Valeurs : AIX 4.2.1 et versions antérieures : Par défaut : 1/8 de la mémoire réelle. Intervalle : jusqu'à 1/2 de la mémoire réelle ou 65536 (64 méga–octets), selon la plus petite valeur.

AIX 4.3.0 : Par défaut : 1/8 de la mémoire réelle. Intervalle : jusqu'à 1/2 de la mémoire réelle ou 131072 (128 méga–octets), selon la plus petite valeur.

AIX 4.3.1 : Par défaut : 1/2 de la mémoire réelle. Intervalle : jusqu'à 1/2 de la mémoire réelle ou 131072 (128 méga–octets), selon la plus petite valeur.

Affichage : **no –a** ou **no –o thewall**

Modification : **no –o thewall=***nouvelle-valeur*

nouvelle-valeur est en ko, et non en octets.

La modification est immédiate pour les nouvelles connexions.

La modification est effective jusqu'à l'amorçage système suivant.

Diagnostics : Aucun.

Optimisation : Augmentez la taille, de préférence à un multiple de 4 ko.

Voir : "Optimisation du pool mbuf", page 9-26.

sockthresh

Objet : Sous AIX Version 4.3.1 ou ultérieure, permet d'éviter la création de nouveaux sockets lorsque presque toute la mémoire réseau est utilisée.

Valeurs : Par défaut : 85% de **thewall**. Intervalle : 1–100

Affichage : **no –a** or **no –o sockthresh**

Modification : **no –o sockthresh=***nouvelle-valeur*

nouvelle–valeur est un pourcentage de **thewall**.

La modification est immédiate.

La modification est effective jusqu'à l'amorçage système suivant.

> **Remarque :** La modification échoue si l'utilisateur tente de sélectionner une nouvelle valeur inférieure à la quantité de mémoire actuellement utilisée.

Diagnostics : Aucun.

Optimisation : Diminuez le pourcentage.

Voir : "Couche socket", page 9-3.

sb_max

Objet : Limite supérieure absolue de la taille des tampons de socket TCP et UDP. Limite **setsockopt()**, **udp_sendspace, udp_recvspace**, **tcp_sendspace** et **tcp_recvspace**.

Valeurs : Par défaut : 65536 Intervalle : N/A.

Affichage : **no –a** ou **no –o sb_max**

Modification : **no –o sb_max=***nouvelle-valeur*

La modification est immédiate pour les nouvelles connexions.

La modification est effective jusqu'à l'amorçage système suivant.

Diagnostics : Aucun.

Optimisation : Augmentez la taille, de préférence à un multiple de 4 096. Doit être environ double de la limite du plus grand tampon de socket.

Voir : "Couche socket", page 9-3.

rfc1323

Objet : 1 indique que **tcp_sendspace** et **tcp_recvspace** peuvent dépasser 64 ko.

Valeurs : Par défaut : 0 Intervalle : 0 ou 1

Affichage : **no –a** ou **no –o rfc1323**

Modification : **no –o rfc1323=***nouvelle-valeur*

La modification est immédiate.

La modification est effective jusqu'à l'amorçage système suivant.

Diagnostics : Aucun.

Optimisation : Effectuez la modification avant de tenter de donner à **tcp_sendspace** et **tcp_recvspace** une valeur supérieure à 64 ko.

Voir : "Couche TCP", page 9-6.

udp_sendspace

Objet : Valeur par défaut de la taille du tampon d'envoi du socket UDP.

Valeurs : Par défaut : 9216 Intervalle : 0 à 65536

Doit être inférieur ou égal à **sb_max**.

Affichage : **no –a** ou **no –o udp_sendspace**

Modification : **no –o udp_sendspace=***nouvelle-valeur*

La modification est immédiate pour les nouvelles connexions.

La modification est effective jusqu'à l'amorçage système suivant.

Diagnostics : Aucun.

Optimisation : Augmentez la taille, de préférence à un multiple de 4 096.

Voir : "Couche socket", page 9-3.

udp_recvspace

Objet Valeur par défaut de la taille du tampon de réception du socket UDP.

Valeurs : Par défaut : 41600 Intervalle : N/A.

Doit être inférieur ou égal à **sb_max**.

Affichage : **no –a** ou **no –o udp_recvspace**

Modification : **no –o udp_recvspace=***nouvelle-valeur*

La modification est immédiate pour les nouvelles connexions.

La modification est effective jusqu'à l'amorçage système suivant.

Diagnostics : *n* non nul dans le compte rendu de **netstat –s** de **udp:** *n* **socket buffer overflows**

Optimisation : Augmentez la taille, de préférence à un multiple de 4 096.

Voir : "Couche socket", page 9-3.

tcp_sendspace

Objet Valeur par défaut de la taille du tampon d'envoi du socket TCP.

Valeurs : Par défaut : 16384 Intervalle : 0 à 64 ko si **rfc1323=0**,

Intervalle : 0 à 4 Go si **rfc1323=1**.

Doit être inférieur ou égal à **sb_max**.

Doit être égal à **tcp_recvspace** et uniforme sur tous les systèmes AIX fréquemment sollicités.

Affichage : **no –a** ou **no –o tcp_sendspace**

Modification : **no –o tcp_sendspace=***nouvelle-valeur*

La modification est immédiate pour les nouvelles connexions.

La modification est effective jusqu'à l'amorçage système suivant.

Diagnostics : Débit faible

Optimisation : Augmentez la taille, de préférence à un multiple de 4 096.

Voir : "Couche socket", page 9-3.

tcp_recvspace

Objet : Valeur par défaut de la taille du tampon de réception du socket TCP.

Valeurs : Par défaut : 16384 Intervalle : 0 à 64 ko si **rfc1323=0**,

Intervalle : 0 à 4 Go si **rfc1323=1**.

Doit être inférieur ou égal à **sb_max**.

Doit être égal à `tcp_sendspace` et uniforme sur tous les systèmes AIX fréquemment sollicités.

Affichage : **no –a** ou **no –o tcp_recvspace**

Modification : **no –o tcp_recvspace=***nouvelle-valeur*

La modification est immédiate pour les nouvelles connexions.

La modification est effective jusqu'à l'amorçage système suivant.

Diagnostics : Débit faible

Optimisation : Augmentez la taille, de préférence à un multiple de 4 096.

Voir : "Couche socket", page 9-3.

ipqmaxlen

Objet : Nombre maximal d'entrées de la file d'entrée IP.

Valeurs : Par défaut : 50 Intervalle : N/A.

Affichage : **no –a** ou **no –o ipqmaxlen**

Modification : **no –o ipqmaxlen =***nouvelle-valeur*

La modification est immédiate.

La modification est effective jusqu'à l'amorçage système suivant.

Diagnostics : Utilisez la commande **crash** pour accéder au compteur de dépassement de la file d'entrée IP.

Optimisation : Augmentez la taille.

Voir : "Optimisation des performances du protocole IP", page 9-23.

xmt_que_size

Objet : Spécifie le nombre maximal de tampons d'envoi susceptibles d'être en file d'attente pour l'unité.

Valeurs : Par défaut : 30 Intervalle : 20 à 150

Affichage : **lsattr –E –l tok0 –a xmt_que_size**

Modification : **ifconfig tr0 detach**

chdev –l tok0 –a xmt_que_size=*nouvelle-valeur*

ifconfig tr0 *nom-hôte* **up**

Le changement reste effectif même après réamorçage du système.

Diagnostics : **netstat –i**

```
Oerr > 0
```

Optimisation : Augmentez la taille. Doit être naturellement définie à 150 sur les systèmes orientés réseau, et notamment les serveurs.

Voir : "Cartes réseau local et pilotes d'unités", page 9-11.

rec_que_size

Objet : (optimisable sous AIX version 3 exclusivement). Spécifie le nombre maximal de tampons de réception qui peuvent être en file d'attente pour l'interface.

Valeurs : Par défaut : 30 Intervalle : 20 à 150

Affichage : **lsattr –E –l tok***n* **–a rec_que_size**

Modification : **ifconfig tr0 detach**

chdev –l tok*n* **–a rec_que_size=***nouvelle-valeur.*

ifconfig tr0 *nom-hôte* **up**

Le changement reste effectif même après réamorçage du système.

Diagnostics : Aucun.

Optimisation : Augmentez la taille. Doit être naturellement définie à 150 sur les systèmes orientés réseau, et notamment les serveurs.

Voir : "Cartes réseau local et pilotes d'unités", page 9-11.

MTU

Objet :	Limite la taille des paquets transmis via le réseau.
Valeurs :	**tr***n* (4 Mo) : Par défaut : 1492 Intervalle : 60 à 3900
	tr*n* (16 Mo) : Par défaut : 1492 Intervalle : 60 à 17960
	en*n*: Par défaut : 1500 Intervalle : 60 à 1500
	fi*n*: Par défaut : 4352 Intervalle : 60 à 4352
	hi*n*: Par défaut : 65536 Intervalle : 60 à 65536
	so*n*: Par défaut : 61428 Intervalle : 60 à 61428
	lo*n*: Par défaut : 1500 (version 3.2.5), 16896 (AIX version 4.1), Intervalle : 60 à 65536
Affichage :	**lsattr –E –l tr***n*
Modification :	**chdev –l tr***n* **–a mtu=***nouvelle-valeur.*
	Ne peut être modifié en cours d'exploitation de l'interface. Tous les systèmes du réseau local (LAN) devant être dotés du même MTU, ils doivent être modifiés simultanément. Le changement reste effectif même après réamorçage.
Diagnostics :	Statistiques de fragmentation de paquets
Optimisation :	Augmentez la taille MTU pour les interfaces en anneau à jeton :
	tr*n* (4 Mo) : 4056
	tr*n* (16 Mo) : 8500
	Pour l'interface en boucle **lo***n* sous la version 3.2.5, augmentez à 16896.
	Pour les autres interfaces, conservez la valeur par défaut.
Voir :	"Cartes réseau local et pilotes d'unités", page 9-11.

Voir aussi

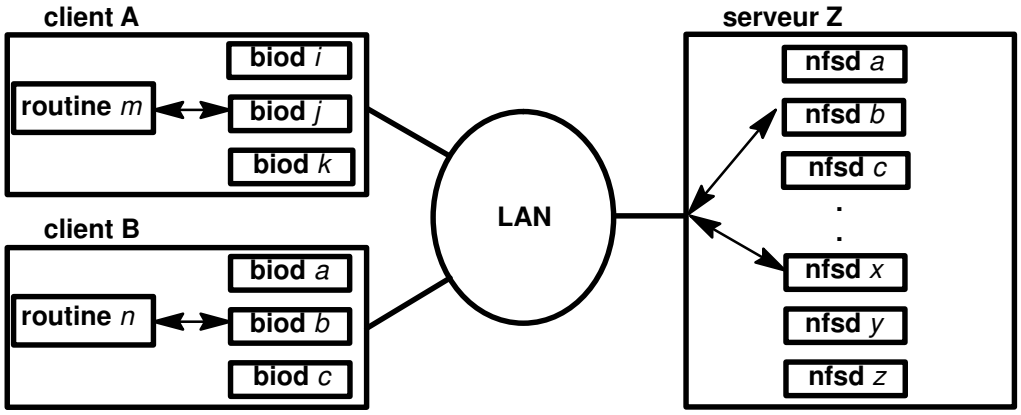
Les commandes **crash**, **ifconfig**, **lsattr**, **netstat** et **no**.

La sous–routine **setsockopt**.

Optimisation de NFS

NFS fournit à des programmes d'un système un accès transparent à des fichiers sur un autre système, par le biais du montage du répertoire distant (via **mount**). Normalement, à l'amorçage du serveur, la commande **exportfs** rend les répertoires disponibles, et les démons gérant le télé-accès (**nfsd**) sont lancés. De même, le **montage** des répertoires distants et l'initialisation du nombre requis de **biod** pour la gestion du télé-accès sont effectués au cours de l'amorçage du système client.

La figure "Interaction client-serveur NFS" illustre la structure du dialogue entre un serveur et des clients NFS. Lorsqu'une routine d'un système client tente de lire ou d'écrire un fichier dans un répertoire monté par NFS, la requête est réacheminée du mécanisme d'E/S normal vers l'un des démons d'E/S par bloc NFS du client (**biod**). Le **biod** envoie la demande au serveur ad hoc, où il est affecté à l'un des démons NFS du serveur (**nfsd**). Pendant le traitement de cette requête, ni le **biod** ni le **nfsd** concerné n'exécutent aucune tâche.



Interaction client-serveur NFS

Nombre de biod et de nfsd requis

Les **biod** et les **nfsd** ne traitant qu'une requête à la fois, et le délai de réponse NFS étant souvent le plus important du total du temps de réponse, il n'est pas souhaitable que des routines soient bloquées suite à l'absence d'un **biod** ou d'un **nfsd**. Voici quelques remarques sur la configuration des démons NFS :

- Augmenter le nombre de démons ne peut compenser un défaut de mémoire ou de puissance du client ou du serveur, ou l'inadéquation de la largeur de bande du disque serveur. Avant de changer le nombre de démons, vérifiez, via les commandes **iostat** et **vmstat**, les niveaux d'utilisation des ressources serveur et client.
- Les démons NFS sont relativement peu coûteux : un **biod** consomme 36 ko de mémoire (9 pages dont 4 fixes), un **nfsd** en consomme 28 ko (7 pages dont 2 fixes). Bien entendu, les pages non fixes ne se trouvent en mémoire réelle que si **nfsd** ou **biod** a été récemment actif. En outre, les **nfsd** AIX inactifs ne consomment pas de temps CPU.
- Toutes les requêtes NFS passent par un **nfsd**, seules les lectures et écritures passent par un **biod**.

Nombre initial de nfsd et de biod

Déterminer le bon nombre de **nfsd** et de **biod** est un processus itératif. Les estimations empiriques ne constituent au mieux qu'un point de départ.

Par défaut, il existe six **biod** sur un client et huit **nfsd** sur un serveur. Ces valeurs par défaut constituent un point de départ satisfaisant pour les petits systèmes, mais devront sûrement être augmentées sur les systèmes client gérant plus de deux utilisateurs ou les serveurs gérant plus de deux clients. Voici quelques conseils :

- Pour chaque client, estimez le nombre maximal de fichiers susceptibles d'être écrits simultanément. Configurez au moins deux **biod** par fichier. Si les fichiers sont volumineux (plus de 32 ko), vous pouvez commencer avec quatre **biod** par fichier pour prendre en charge les activités de lecture anticipée ou d'écriture différée. Il n'est pas rare de mobiliser cinq **biod** pour écrire un fichier volumineux.
- Sur chaque serveur, commencez par configurer autant de **nfsd** que la somme des **biod** configurés sur les clients pour gérer les fichiers *à partir de ce serveur*. Ajoutez 20 % pour les requêtes NFS autres que de lecture/écriture.
- Si les stations client sont rapides, mais connectées à un serveur plus lent, vous devrez peut-être limiter le taux de génération des requêtes NFS émises par les clients. Le mieux est de réduire le nombre de **biod** sur les clients, en tenant compte de l'importance relative de la charge et du temps de réponse de chaque client.

Optimisation du nombre de nfsd et de biod

Après avoir déterminé ou modifié le nombre initial de **biod** et de **nfsd** :

- Contrôlez à nouveau les systèmes concernés, via **vmstat** et **iostat**, à la recherche de saturation au niveau des E/S ou de la CPU. Si le serveur est saturé, vous devez diminuer sa charge ou augmenter sa puissance, ou les deux.
- Lancez **netstat –s** pour déterminer si un système n'est pas confronté à un dépassement de tampon de socket UDP. Dans l'affirmative, lancez **no –a** pour vérifier que les conseils indiqués à "Optimisation des autres couches pour améliorer les performances NFS", page 9-39 ont bien été suivis. Si oui, et que le système n'est pas saturé, augmentez le nombre de **biod** ou de **nfsd**.

La commande **chnfs** permet de modifier le nombre de **nfsd** et de **biod**. Ainsi, pour définir à 10 le nombre de **nfsd** sur un serveur, immédiatement et à chaque réamorçage, entrez :

```
# chnfs –n 10
```

Pour définir temporairement à 8 le nombre de **biod** sur un client, sans pérenniser le changement (perdu à l'amorçage suivant), entrez :

```
# chnfs –N –b 8
```

Pour définir à 9 le nombre de **biod** et de **nfsd**, le changement ne prenant effet qu'à l'amorçage (IPL) suivant du système, entrez :

```
# chnfs –I –b 9 –n 9
```

Dans le cas, extrême, d'un client provoquant une surcharge du serveur, vous devrez peut-être ramener le client à un seul **biod**. Pour ce faire, lancez la commande :

```
# stopsrc –s biod
```

Le client ne dispose plus que du **biod** kproc.

Performances et montages logiciels et matériels de NFS

Un des choix que vous serez amenés à faire lorsque vous configurez des répertoires montés via NFS est de décider si le montage est logiciel ou matériel. Si, après un montage réussi, l'accès à un répertoire monté logiquement génère une erreur (dépassement de délai, le plus souvent), l'erreur est immmédiatement reportée au programme qui a demandé le télé-accès. Si une erreur est générée lors de l'accès à un répertoire monté physiquement, NFS relance l'opération.

Une erreur persistante au niveau de l'accès à un répertoire monté physiquement peut entraîner une nette dégradation des performances, car le nombre de réessais par défaut (1 000) et le délai par défaut (0,7 seconde), combinés à un algorithme qui augmente le délai imparti à chaque nouvelle tentative, conduit NFS à relancer l'opération pratiquement indéfiniment (tant qu'elle n'aboutit pas).

Il est techniquement possible de réduire le nombre de relances, ou d'augmenter la valeur du délai, ou les deux, via les options de la commande **mount**. Malheureusement, modifier

substantiellement ces valeurs (pour supprimer effectivement le problème de performance) peut entraîner l'indication inutile d'erreurs matérielles. Mieux vaut monter physiquement les répertoires avec l'option **intr**, qui permet à l'utilisateur d'interrompre à partir du clavier un processus se trouvant dans une boucle de relance.

Bien que le montage logiciel des répertoires permette une détection plus rapide des erreurs, le risque est grand que des données soient endommagées. Pour les répertoires de lecture/écriture, optez de préférence pour un montage physique.

Optimisation des retransmissions

Parallèlement au choix du type de montage (physique ou logique), se pose la question du délai imparti dans une configuration donnée. Si le serveur est très chargé, qu'il est séparé du client par un ou plusieurs ponts ou passerelles, ou qu'il est connecté au client via un WAN, le délai imparti par défaut peut se révéler peu réaliste. Dans ce cas, serveur et client se retrouveront surchargés par des retransmissions inutiles. Si, par exemple :

```
$ nfsstat -cr
```

génère un nombre significatif (> 5 % du total) de `timeout` s et de `badxid` s, il convient d'augmenter la valeur du paramètre **timeo** via :

```
# smit chnfsmnt
```

Identifiez le répertoire à modifier, et entrez une nouvelle valeur sur la ligne "NFS TIMEOUT. In tenths of a second". Pour un trafic LAN-LAN via un pont, essayez 50 (dixièmes de seconde). Pour une connexion WAN (longue distance), essayez 200. Après au moins une journée, vérifiez les statitisques NFS. Si elles indiquent toujours un nombre excessif de retransmissions, augmentez encore **timeo** de 50 % et réessayez. Vérifiez également la charge du serveur et des ponts et passerelles concernés, pour déterminer si un élément est saturé par un trafic autre.

Optimisation du cache d'attribut de fichier NFS

NFS tient à jour, sur chaque système client, un cache des attributs des répertoires et des fichiers auxquels un accès a récemment été effectué. Cinq paramètres, définissables dans le fichier /**etc**/**filesystems**, contrôlent la durée de maintien d'une entrée dans le cache. Il s'agit de :

actimeo	Durée absolue pendant laquelle les entrées fichier et répertoire sont conservées dans le cache des attributs de fichier après une mise à jour. Si elle est spécifiée, elle prévaut sur les valeurs ***min** et ***max** ci-après, leur donnant effectivement la valeur **actimeo**.
acregmin	Durée minimale pendant laquelle les entrées fichier sont conservées après une mise à jour. Valeur par défaut : 3 secondes.
acregmax	Durée maximale pendant laquelle les entrées fichier sont conservées après une mise à jour. Valeur par défaut : 60 secondes.
acdirmin	Durée minimale pendant laquelle les entrées répertoire sont conservées après une mise à jour. Valeur par défaut : 30 secondes.
acdirmax	Durée maximale pendant laquelle les entrées répertoire sont conservées après une mise à jour. Valeur par défaut : 60 secondes.

Chaque fois qu'un fichier ou un répertoire est mis à jour, sa suppression est différée d'au moins **acregmin** ou **acdirmin** secondes. S'il s'agit d'une deuxième ou nième mise à jour, l'entrée est conservée pendant une durée au moins égale à l'intervalle entre les deux dernières mises à jour, sous réserve qu'elle reste inférieure à **acregmax** ou **acdirmax** secondes.

Désactivation du support ACL NFS inutilisé

Si votre charge de travail n'exploite pas le support ACL de NFS sur un système de fichiers monté, vous pouvez réduire la charge du serveur et du client en spécifiant :

```
options = noacl
```

dans la strophe /**etc**/**filesystems** du client, pour ce système de fichiers.

Optimisation de la mise en cache des données NFS

NFS ne propose pas de fonction de mise en cache de données, mais le gestionnaire de mémoire virtuel (VMM) d'AIX cache les pages de données NFS exactement comme les pages de données disque. Si un système est prioritairement un serveur NFS dédié, il peut être souhaitable d'autoriser le VMM à utiliser autant de mémoire que nécessaire pour cacher les données. Pour ce faire, donnez la valeur 100 % au paramètre **maxperm**, qui contrôle le pourcentage maximal de mémoire occupé par les pages de fichier, via la commande :

```
# vmtune -P 100
```

La même technique peut s'appliquer aux clients NFS, mais n'a de sens que si leur charge de travail ne requiert que très peu de pages de segments de travail.

Optimisation des autres couches pour améliorer les performances NFS

NFS passe par UDP pour effectuer ses E/S réseau. Assurez-vous que les techniques d'optimisation décrites à "Optimisation de TCP et UDP", page 9-12 et "Optimisation du pool mbuf", page 9-26 ont été appliquées. Vous devez, notamment :

- Vérifiez que les files de transmission et de réception de la carte LAN sont définies à leur maximum (150).
- Passez la taille maximale du tampon du socket (**sb_max**) à au moins 131 072. Si la taille du MTU est inférieure à 4 096 octets, donnez à **sb_max** une valeur supérieure ou égale à 262 144. Donnez également à la taille des tampons de socket UDP (**udp_sendspace** et **udp_recvspace**) la valeur 131 072.
- Si possible, augmentez la taille du MTU sur le réseau local. Sur un anneau à jeton 16 Mo, par exemple, passer de la taille par défaut (1 492 octets) à 8 500 octets permet de transmettre sans fragmentation une requête de lecture ou d'écriture complète NFS de 8 ko. Cette opération optimise également l'exploitation de l'espace mbuf, réduisant les risques de surcharges.

Augmentation de la taille du tampon du socket NFS

Au cours de l'optimisation d'UDP, vous pouvez découvrir que la commande :

```
$ netstat -s
```

indique un nombre non négligeable de débordements du tampon du socket UDP. Comme pour l'optimisation classique d'UDP, vous devez augmenter la valeur de **sb_max**. Vous devez également augmenter la valeur de **nfs_chars**, qui spécifie la taille du tampon, du socket NFS. La séquence :

```
# no -o sb_max=131072
# nfso -o nfs_chars=130000
# stopsrc -s nfsd
# startsrc -s nfsd
```

donne à **sb_max** une valeur d'au moins 100 octets supérieure à la valeur souhaitée de **nfs_chars**, donne à **nfs_chars** la valeur 130 972, puis arrête et relance les **nfsd** pour que les nouvelles valeurs entrent en vigueur. Si vous constatez que cette modification améliore les performances, insérez les commandes **no** et **nfso** dans /**etc**/**rc.nfs**, juste avant la commande **startsrc** qui lance les **nfsd**.

Configuration du disque du serveur NFS

Pour les serveurs NFS qui ont à faire face à un niveau élevé d'activité d'écriture, vous avez intérêt à configurer le volume logique journal sur un volume physique distinct de celui des données. Reportez-vous à "Préinstallation du disque", page 4-24.

Accélérateurs matériels

Prestoserve

Le but du produit Prestoserve est de réduire le temps de latence des écritures NFS, en proposant une méthode plus rapide que les E/S disque pour satisfaire l'impératif NFS d'écritures synchrones. Il fournit de la RAM non volatile (NVRAM) dans laquelle NFS peut écrire des données. Les données sont alors considérées comme "en sécurité" et NFS peut autoriser le client à poursuivre. Les données sont écrites ultérieurement sur un disque, en fonction de la disponibilité des unités. Enfin, il est impossible de dépasser la largeur de bande du disque, mais dans la mesure où l'essentiel du trafic NFS est en rafales, Prestoserve permet d'aplanir la charge de travail sur le disque, avec des gains de performance parfois considérables.

Interphase Network Coprocessor

Ce produit gère le traitement du protocole NFS sur les réseaux Ethernet, réduisant la charge de la CPU. Le traitement du protocole NFS est particulièrement onéreux sur les réseaux Ethernet, car les blocs NFS doivent être fractionnés pour correspondre à la taille de MTU maximale (1 500 octets) de Ethernet.

Du bon usage de NFS

Le plus souvent, l'exploitation incorrecte de NFS est due au fait que les utilisateurs oublient que les fichiers auxquels ils accèdent se trouvent à l'autre extrémité d'une voie de communication coûteuse. Quelques exemples (réels) :

- Une application COBOL sous un système AIX effectuant des mises à jour aléatoires d'un fichier d'inventaire monté via NFS – prenant en charge une application de registre de caisse de détail en temps réel.
- Environnement de développement dans lequel un répertoire de code source sur chaque système a été monté via NFS sur tous les autres systèmes dans l'environnement – les développeurs se connectant à n'importe quel système pour modifier leurs programmes et les compiler. Cette structure garantit quasi-inévitablement que le code source de toutes les compilations est importé depuis des systèmes distants, et réciproquement pour le résultat des compilations.
- Exécution de la commande **ld** sur un système pour transformer les fichiers **.o** d'un répertoire monté via NFS en fichier **a.out** dans le même répertoire.

Il peut être argué qu'il s'agit là d'une exploitation légitime de la transparence offerte par NFS. Sans doute, mais ces pratiques coûtent du temps processeur, consomment de la largeur de bande et augmentent les temps de réponse. Lorsqu'un système est configuré de sorte que l'accès à NFS fasse partie des opération standard, les concepteurs de cette configuration doivent pouvoir défendre leur choix si coûteux par de substantiels avantages techniques ou commerciaux :

- Disposer de toutes les données ou du code source sur un serveur, et non sur des postes individuels améliore le contrôle du code source et facilite la centralisation des sauvegardes.
- Plusieurs systèmes ont accès aux mêmes données, rendant un serveur dédié plus efficace qu'un ou plusieurs systèmes combinant les rôles de client et de serveur.

Service des stations de travail sans disque

Potentiellement, les systèmes sans disque offrent une excellente puissance de traitement couplée à un faible coût, une émission sonore réduite, peu de besoin d'espace et une gestion centralisée des données. Aussi tentants que semblent ces avantages, les stations sans disque ne sont pas forcément l'idéal pour tous les bureaux. Cette section a pour but d'éclairer certains aspects du fonctionnement des stations sans disque AIX : types de charges présentées aux serveurs et performances de différents types de programmes. L'essentiel de ce qui traite de NFS ici s'applique également aux stations avec disques.

Cette section traite des points suivants :

- "Particularités d'un système sans disque"
- "Remarques sur NFS"
- "Exécution d'un programme sur une station de travail sans disque"
- "Pagination"
- "Ressources requises pour une station de travail sans disque"
- "Optimisation des performances"
- "Performances des commandes"
- "Etude de cas 1 – Environnement de bureau"
- "Etude de cas 2 – Environnement de développement de logiciels"

Particularités d'un système sans disque

Dans un système équipé de disques locaux (appelé système *à disque*), le système d'exploitation et les programmes requis pour la plupart des fonctions de base se trouvent sur un ou plusieurs disques locaux. Au lancement du système, le système d'exploitation est chargé à partir du disque local. Lorsque le système est complètement opérationnel, les fichiers accessibles aux utilisateurs se trouvent également sur disque local. Les disques locaux sont gérés par le système JFS (système de fichiers journalisé).

Dans un système sans disque, il faut amorcer le système d'exploitation à partir d'un serveur via un code d'amorçage se trouvant dans la ROM de la machine sans disque.
Le chargement a lieu sur un réseau local : Ethernet ou anneau à jeton. Lorsque le système est complètement opérationnel, les fichiers accessibles aux utilisateurs se trouvent sur les disques d'un ou de plusieurs systèmes serveur.

NFS (Network File System) est le principal mécanisme utilisé par les stations de travail sans disque pour accéder aux fichiers. Avec NFS, les fichiers distants semblent se trouver sur le système sans disque. NFS n'est pas propre aux systèmes sans disque. Les systèmes à disque peuvent également monter des systèmes de fichiers distants. Les systèmes sans disque, ou à disque mais dépendant de serveurs pour les fichiers, sont généralement appelés des *clients*.

Normalement, plusieurs clients sans disque sont reliés à chaque serveur, il y a donc concurrence pour l'accès aux ressources serveur. La différence de performance entre deux systèmes identiques, l'un doté de disque et l'autre non, dépend des systèmes de fichiers (NFS ou JFS), de la vitesse du réseau et des ressources du serveur.

Remarques sur NFS

NFS (Network File System) multiplie les accès client aux données télémontées. Il fournit des primitives pour les fonctions de base (création, lecture, écriture, suppression, etc. des systèmes de fichiers). NFS assure également des opérations sur les répertoires (suppression, lecture et définition des attributs, consultation du chemin d'accès, etc.).

Le protocole utilisé par NFS est "sans état" : aucune requête au serveur ne dépend d'une requête antérieure. Ce qui ajoute à la solidité du protocole. Mais qui introduit également des

problèmes de performances. Considérons le cas de l'écriture d'un fichier. Pendant cette écriture, les données modifiées se trouvent soit dans la mémoire client, soit sur le serveur. Le protocole NFS requiert que les données écrites du client vers le serveur soient consignées dans un espace de stockage non volatil (le disque généralement) avant que l'écriture ne soit considérée terminée. Ainsi, en cas de défaillance du serveur, les données écrites par le client peuvent être recouvrées après réinitialisation du système. Les données en cours d'écriture, non consignées sur le disque sont réécrites par le client sur le serveur jusqu'à ce que l'opération aboutisse. NFS n'autorisant pas la mise en tampon des écritures dans le serveur, chaque écriture NFS requiert une ou plusieurs écritures disque synchrones. Par exemple, si un nouveau fichier de 1 octet est écrit par le client, l'exécution de cette écriture entraîne trois E/S disque sur le serveur : la première sont les données elles-même. La seconde est l'enregistrement du journal, une fonction JFS destinée à maintenir d'intégrité du système de fichiers. La troisième est un cliché des données d'allocation de fichier. Les écritures sur disque étant limitées à 50 à 100 par seconde, le total des sorties d'écriture est limité par le nombre et le type de disque du serveur.

Les requêtes de lecture/écriture émises par les clients AIX sont de 4 096 ou 8 192 octets. Elles requièrent généralement plus de ressources serveur que les autres types de requêtes.

Les fichiers distants et leur attributs pouvant se trouver dans la mémoire cache du client, le protocole NFS fournit un outil permettant de vérifier que la version client des informations sur le système de fichiers est à jour. Par exemple, si un fichier de 1 octet est en cours de lecture, le fichier de données sera en mémoire cache tant que l'espace qu'il occupe sur le client n'est pas réquisitionné par une autre activité. Si un programme du client relit le fichier plus tard, le client assure que les données de la copie locale du fichier sont actuelles. Cette opération est effectuée par le biais d'un appel Get Attribute au serveur pour déterminer si le fichier a été modifié depuis sa dernière lecture.

La *résolution des noms de chemin* est le processus de parcours de l'arborescence des répertoires jusqu'au fichier. Par exemple, ouvrir le fichier `/u/x/y/z` requiert normalement d'examiner `/u, x, y` et `z`, dans cet ordre. Si un des éléments du chemin n'existe pas, le fichier ne peut exister sous son nom. Un des caches NFS est dédié aux noms fréquemment utilisés, réduisant d'autant le nombre de requêtes acheminées vers le serveur.

Le serveur reçoit évidemment un panachage de requêtes de lecture, d'écriture et autres, au cours d'un intevalle de temps quelconque. Ce panachage est quasi-impossible à prévoir. Les travaux qui impliquent de fréquents déplacements de fichiers volumineux entraînent une prédominance des requêtes de lecture/écriture. La prise en charge de plusieurs stations sans disque tend à générer une proportion plus importante de petites requêtes NFS – mais tout dépend beaucoup de la charge de travail.

Exécution d'un programme sur une station de travail sans disque

Pour mieux appréhender le flux des requêtes NFS sur un client sans disque, étudions l'exécution du shell Korn du programme trivial C :

```
#include <stdio.h>
main()
{
printf("Ceci est un programme test\n");
}
```

Le programme est compilé, générant l'exécutable `a.out`. Si la variable d'environnement **PATH** est `/usr/bin:/usr/bin/X11:`(le point représentant le répertoire de travail courant se trouve à la fin du chemin) et que la commande `a.out` est entrée sur la ligne de commande, voici la séquence exécutée :

	type de requête	composant	octets envoyés et reçus
1	NFS_LOOKUP	usr (appelé par statx)	(send 178, rcv 70)
2	NFS_LOOKUP	bin	

3	NFS_LOOKUP	a.out	(non trouvé)
4	NFS_LOOKUP	usr (appelé par statx)	
5	NFS_LOOKUP	bin	
6	NFS_LOOKUP	X11	(send 174, rcv 156)
7	NFS_LOOKUP	a.out (non trouvé)	(send 174, rcv 70)
8	NFS_LOOKUP	. (appelé par statx)	(send 174, rcv 156)
9	NFS_LOOKUP	.	
10	NFS_LOOKUP	a.out	(send 178, rcv 156)
11	NFS_LOOKUP	. (appelé par accessx)	
12	NFS_LOOKUP	a.out	
13	NFS_GETATTR	a.out	
14	NFS_LOOKUP	.	
15	NFS_LOOKUP	a.out	(send 170, rcv 104, send 190, rcv 168)
16	fork		
17	exec		
18	NFS_LOOKUP	usr	
19	NFS_LOOKUP	bin	
20	NFS_LOOKUP	a.out (non trouvé)	(send 178, rcv 70)
21	NFS_LOOKUP	usr	
22	NFS_LOOKUP	bin	
23	NFS_LOOKUP	X11	
24	NFS_LOOKUP	a.out (non trouvé)	(send 178, rcv 70)
25	NFS_LOOKUP	.	
26	NFS_LOOKUP	a.out	
27	NFS_OPEN		(send 166, rcv 138)
28	NFS_GETATTR	a.out	
29	NFS_ACCESS		(send 170, rcv 104, send 190, rcv 182)
30	NFS_GETATTR	a.out	
31	FS_GETATTR	a.out	
32	NFS_READ	a.out (exécutable Read)	(send 178, rcv 1514, rcv 1514, rcv 84)
33	NFS_GETATTR	a.out	
34	NFS_LOOKUP	usr (Access library)	
35	NFS_LOOKUP	lib	
36	NFS_LOOKUP	libc.a	
37	NFS_READLINK	libc.a	(send 166, rcv 80)
38	NFS_LOOKUP	usr	
39	NFS_LOOKUP	ccs	
40	NFS_LOOKUP	lib	

41	NFS_LOOKUP	libc.a	
42	NFS_OPEN	libc.a	(send 166, rcv 124)
43	NFS_GETATTR	libc.a	
44	NFS_ACCESS	libc.a	(send 170, rcv 104, send 190, rcv 178)
45	NFS_GETATTR	libc.a	
46	NFS_GETATTR	libc.a	
47	NFS_CLOSE	libc.a	
48	_exit		

Avec une autre variable **PATH**, la série d'opérations NFS serait différente. Par exemple, avec la variable **PATH** `.:/usr/bin:/usr/bin/X11:` , le programme `a.out` aurait été trouvée plus rapidement. A l'inverse, avec ce **PATH**, la plupart des commandes auraient été exécutées plus lentement, car elles résident presque toutes dans `/usr/bin`. Un autre moyen rapide d'exécuter le programme est d'entrer `./a.out`, dans la mesure où il est inutile de parcourir l'exécutable (la résolution de bibliothèque est néanmoins toujours requise). Ajouter un nombre important de répertoires peu utilisés au chemin (**PATH**) ralentit l'exécution des commandes. Ceci s'applique à tous les environnements, mais est particulièrement significatif dans les environnements sans disque.

Un autre facteur à prendre en compte lors du développement de programmes est la minimisation du nombre de bibliothèques référencées. Il est évident que plus il y a de bibliothèques à charger, moins l'exécution du programme est rapide. La variable d'environnement **LIBPATH** peut également affecter la vitesse de chargement du programme. Soyez donc prudent lorsque vous l'utilisez.

Le support NLS (National Language Support) peut aussi avoir une incidence sur l'exécution des programmes. Dans l'exemple précédent, l'exécution avait lieu dans l'environnement local "C", le plus efficace. L'exécution dans d'autres environnements peut induire une charge supplémentaire lors de l'accès aux catalogues de messages.

A première vue, l'activité NFS pour un petit programme semble disproportionnée. En fait, les performances de l'exemple restent dans des normes acceptables. N'oubliez pas que la résolution de chemin pour l'accès aux fichiers est également effectuée pour les systèmes de fichiers JFS : le nombre total d'opérations est donc similaire. Le cache NFS assure que les opérations NFS ne généreront pas toutes de trafic réseau. Au total, les délais de latence des opérations réseau restent généralement courts, et le cumul des délais pour les commandes peu élevé – sauf si le serveur est lui-même surchargé.

Pagination

Les systèmes sans disque AIX effectuent la pagination via le protocole NFS. La pagination est le processus par lequel l'espace de travail (variables du programme, par exemple) peut être écrit et lu sur disque. Cette opération a lieu lorsque la somme des besoins en mémoire des processus exécutés sur un système est supérieure à la mémoire système. (Reportez-vous à "Performances du gestionnaire de la mémoire virtuelle (VMM)", page 2-5.)

Les avantages de la pagination sont mitigés. Elle permet certes d'éviter de surcharger la mémoire, mais généralement aux dépens des performances. En fait, seule une faible dose de pagination maintient des temps de réponse acceptables dans un environnement de stations de travail.

Dans l'environnement sans disqsue, la pagination est particulièrement lente. Ceci est dû au protocole NFS qui force les écritures sur disque. En pratique, on peut considérer qu'au mieux, l'écriture d'une page prendra deux à trois fois plus de temps que sur un système à disque. Compte tenu de ce fait, il est important que les systèmes sans disque disposent de suffisamment de mémoire pour que les applications exécutées n'aient pas besoin de recourir à la pagination. (Voir "Programmes à CPU limitée".)

Les produits bureautiques AIXwindows favorisent les attitudes conduisant à des périodes de pagination intense sur des systèmes dépourvus de mémoire suffisante. Par exemple, un utilisateur peut lancer simultanément deux programmes : un tableur conséquent dans une fenêtre et une base de données dans une autre. Supposons qu'il mette à jour une feuille de calcul, attende le résultat puis passe à la fenêtre base de données pour interroger la base. Bien que le tableur soit inactif, il occupe un espace non négligeable. Interroger une base de données requiert également beaucoup d'espace. Sauf si la mémoire réelle est suffisante pour contenir ces deux programmes, les pages de mémoire virtuelle du tableur sont paginées vers l'extérieur et la base de données chargée. A l'interaction suivante de l'utilisateur avec le tableur, la mémoire occupée par la base de données doit être paginée, et le tableur rechargé. En clair, les limites de cette situation sont déterminées par la fréquence de passage d'une fenêtre à l'autre et par la charge qui incombe alors au serveur.

Ressources requises pour les stations de travail sans disque

Plusieurs services AIX permettent de mesurer la charge client-serveur. Le nombre de requêtes NFS traitées par un système est accessible via **nfsstat**. Cette commande décompte en détail les requêtes NFS par catégorie. La commande **netstat** analyse le décompte de paquets et d'octets transférés à une unité du réseau. La commande **iostat** détaille l'utilisation du processeur et du disque – paramètres utiles pour les mesures sur les systèmes serveur. Enfin, l'utilitaire de suivi AIX permet de rassembler des données précises sur les performances.

Planifier la capacité de réseaux sans disque est une tâche souvent compliquée par l'intermittence des requêtes d'E/S des clients – courtes périodes enregistrant des pointes de requêtes entrecoupées de longues périodes où le taux de requête est très bas.
Ce phénomène est courant sur les systèmes où les applications sont principalement pilotées par les utilisateurs.

La capacité, c'est-à-dire le nombre de clients pris en charge par un serveur et un réseau pour une charge de travail donnée, est déterminée par les statistiques des requêtes et les besoins des utilisateurs finals. Quelques questions doivent être posées.

- Quelle est la fréquence réelle d'exécution de ce travail ? Normalement, un utilisateur passe l'essentiel de son temps à des tâches autres que la compiltaion et l'édition de liens de programmes. Supposer que tous les autres utilisateurs passent tout leur temps en interaction avec leur station de travail à une vitesse maximale conduit à une surestimation du nombre d'utilisateurs susceptibles d'être pris en charge.
- Quelle est l'utilisation moyenne acceptable du réseau ? Pour les réseaux Ethernet, le pourcentage est de 30 à 60 %, selon le site.
- Quelle est la probabilité qu'un nombre important de clients se trouvent simultanément dans une période d'utilisation de pointe du réseau ? Dans une telle période, les temps de réponse augmentent. Quelle est la fréquence de ces pointes simultanées ? Quelle est leur durée ?

Il est parfois intéressant de télé-exécuter les applications volumineuses. Exécuter l'application sur un serveur, via une fenêtre distante, donne au client l'avantage de bénéficier de la mémoire du serveur et permet à plusieurs instances de l'application appelées par différents clients de partager des pages de code et des fichiers document.
Si l'application est exécutée sur le client, les passages d'une fenêtre à l'autre, tels que décrits précédemment, ont une incidence catastrophique sur les temps de réponse.
D'autres tâches, impliquant par exemple de vastes et intenses opérations **make** ou **cp**, peuvent également bénéficier du déplacement de l'application à proximité des disque fixes.

Lors de la configuration des réseaux et des serveurs pour des clients sans disque, il convient d'effectuer dès que possible les mesures relatives aux applications. N'oubliez pas de mesurer l'utilisation du processeur du serveur ainsi que celle du disque. Ces réseaux sont plus susceptibles de présenter des goulots d'étranglement que les réseaux Ethernet ou en anneau à jeton 16 Mo.

Optimisation des performances

La capacité d'une configuration client/serveur doit être pensée en termes de fourniture et de demande. La fourniture des ressources dépend du type du réseau et de la configuration du serveur. La demande est la somme de tous les besoins client vis-à-vis du serveur. Si une configuration génère des performances inacceptables, modifier la demande client ou augmenter la fourniture des ressources serveur ne peut qu'améliorer la situation.

Le taux d'utilisation correspond au pourcentage de temps pendant lequel une unité est mobilisée. Les unités dont le taux d'utilisation est supérieur à 70 % constateront rapidement un allongement des temps de réponse, les requêtes entrantes devant attendre la fin du traitement de la requête précédente. Les taux d'utilisation acceptables maximum sont un compromis entre temps de réponse et débit. Sur un système interactif, le taux d'utilisation des unités ne doit pas excéder 70 à 80 % pour des temps de réponse acceptables. Les systèmes en différé, où le débit des flux de travaux multiples est important, peuvent "tourner" à quasi 100 %. Dans des systèmes mixtes (interactifs et différés), il convient bien entendu de ne pas trop grever les temps de réponse.

Optimisation du client

Plusieurs opérations peuvent concourir à l'optimisation d'un client :

- Ajout de mémoire client
- Augmentation du nombre des démons **biod** du client NFS
- Modification de la configuration réseau du client
- Ajout d'un disque à la configuration client

Si la mémoire d'un client est insuffisante, l'espace de travail est paginé. Ce qui peut être détecté via la commande **vmstat –s**. Si l'espace de travail d'un client est en permanence paginé, ajouter de la mémoire ne peut qu'améliorer ses performances.

Le nombre de démons d'E/S par bloc (**biod**) configurés sur un client limite le nombre de requêtes de lecture et d'écriture NFS en suspens. Sur un système sans disque sur lequel NFS n'a pas été explicitement activé, seuls quelques **biod** sont disponibles. Si NFS est activé, ce nombre augmente. Normalement, le nombre de **biod** disponibles par défaut avec NFS activé est suffisant pour une station sans disque.

Les pilotes d'unité Ethernet et en anneau à jeton sont assortis de paramètres qui définissent la taille des files de transmission et de réception pour l'unité. Ces paramètres peuvent avoir une incidence sur les performances client. Voir "Optimisation des autres couches pour améliorer les performances NFS", page 9-39.

Ajouter un disque à une station sans disque n'est pas une hérésie : en fait, des études marketing ont montré que les systèmes sans disque sont généralement équipés d'un disque dans l'année qui suit leur achat. Ajouter un disque ne met pas en cause le principal avantage des systèmes sans disque – la maintenance centralisée des fichiers. Le disque peut être réservé à la pagination. On parle alors d'un système *sans données.* D'autres combinaisons sont possibles. Un disque peut par exemple contenir de l'espace de pagination et de l'espace pour les fichiers temporaires.

Optimisation du réseau

La largeur de bande d'un réseau est, nominalement, de 10 megabits/seconde. Dans la pratique, la concurrence entre utilisateurs Ethernet rend impossible l'exploitation de l'intégralité de la largeur de bande. Sachant qu'un disque SCSI Bull peut fournir jusqu'à 32 megabits/seconde, il est inquiétant de voir de nombreux clients partager le même quart de largeur de bande d'un disque. Cette comparaison ne vaut, toutefois, que pour les applications qui effectuent des E/S disque séquentielles – ce qui n'est pas le plus courant, les E/S aléatoires (limitées en temps de recherche et délai de rotation) étant nettement plus fréquentes. La plupart des disques SCSI ayant un débit soutenu de 50 à 85 opérations d'E/S aléatoires par seconde, le taux effectif d'E/S aléatoires d'un disque est de 2 - 3 mégabits/seconde. Ainsi, une largeur de bande Ethernet est sensiblement équivalente

à deux disques effectuant des E/S aléatoires. Il convient d'en tirer une leçon : les applications effectuant des E/S séquentielles sur des fichiers volumineux doivent être exécutées sur le système auquel est rattaché le disque, et non sur une station sans disque.

Bien que le MTU (Maximum Transfer Unit) d'un réseau local (LAN) puisse être modifé via SMIT, les stations sans disque sont limitées à leur valeur par défaut.

Optimisation du serveur

La configuration du serveur fait intervenir :

- CPU du serveur
- Configuration du disque du serveur
- Configuration du NFS du serveur
- Configuration de la mémoire du serveur
- Configuration du réseau du serveur

La puissance de traitement de la CPU du serveur est significative car toutes les requêtes serveur font appel au service CPU. Généralement, le traitement CPU requis par les requêtes de lecture/écriture est sensiblement supérieur à celui requis par les autres requêtes.

La configuration du disque du serveur est généralement le premier goulot d'étranglement rencontré. Une technique d'optimisation évidente consiste à équilibrer les E/S disque de sorte qu'aucune utilisation disque ne soit largement supérieure à une autre. Une autre technique est de maximiser le nombre de disques. Par exemple, deux disques de 400 Mo offrent près de deux fois plus de possibilités d'E/S aléatoires par seconde qu'un seul disque de 857 Mo. En outre, avec AIX, il est possible de placer un historique du journal sur une autre unité. Cettet opération améliore la séquence d'écritures NFS multiple comme suit :

- Ecriture des données sur un disque fichier
- Ecriture de l'historique du journal sur le disque journal (pas de recherche de disque)
- Ecriture des données d'affectation de fichier sur disque fichier (recherche de disque peu importante)

De par l'absence du journal sur le disque fichier, une ou deux recherches potentiellement longues sur le disque sont évitées. (Si le fichier et l'historique du journal se trouvaient sur un même disque peu chargé, l'accesseur passeraient continuellement de la zone de fichier à la zone de l'historique de journal, et vice-versa.)

Le nombre d'instances du démon NFS (**nfsd**) exécutées sur le serveur limite le nombre de requêtes NFS exécutables concuremment par le serveur. Le nombre de **nfsd** par défaut n'est que de 8, ce qui n'est généralement suffisant que pour les serveurs d'entrée de gamme. Le nombre de **nfsd** lancés à chaque amorçage peut être modifié via **smit nfs** (**Network File System (NFS)** –> **Configuration NFS sur ce système**).

La taille de la mémoire du serveur n'est significative que pour les opérations de lecture NFS. Dans la mesure où les écritures ne peuvent être placées en mémoire cache, la taille de la mémoire est sans incidence sur les performances relatives à l'écriture. D'autre part, en supposant que certains fichiers sont utilisés à répétition, plus la mémoire du serveur est élevée, plus grande est la probabilité qu'une lecture puisse être satisfaite par la mémoire, en évitant une E/S disque. Gagner une E/S disque réduit le taux d'utilisation du disque, améliore les temps de réponse pour les lectures et diminue le taux d'utilisation de la CPU du serveur. Vous disposez de la commande **iostat** pour étudier l'activité du disque.
Les indices suivants révélent qu'un supplément de mémoire peut améliorer les performances du serveur :

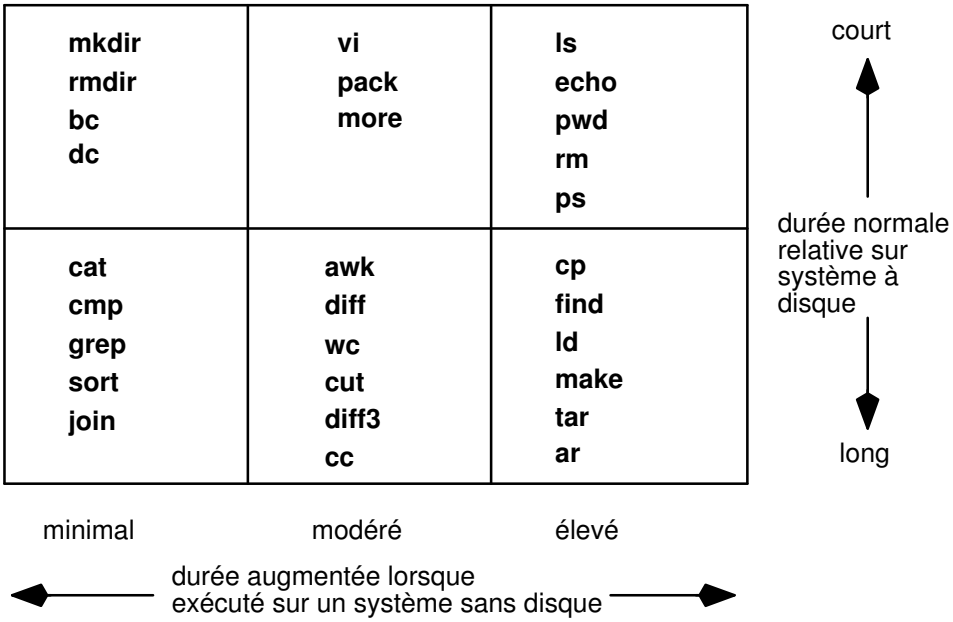
- Un ou plusieurs pilotes de disque opèrent presque à la limite de leurs capacités (40 à 85 E/S aléatoires par seconde). Voir "Préinstallation du disque", page 4-24.

- Sur une période de quelques minutes ou plus, le nombre d'octets lus est sensiblement supérieur au nombre d'octets écrits.

Comme le client, les pilotes d'unité Ethernet et en anneau à jeton sont limités quant au nombre de tampons disponibles pour l'envoi des données. Voir "Optimisation des autres couches pour améliorer les performances NFS", page 9-39.

Performance des commandes

Les commandes AIX subissent le même type d'actions que celles constatées lors de l'exécution d'un programme trivial (voir "Exécution d'un programme sur une station de travail sans disque", page 9-42). Le comportement des commandes peut être déduit du type et du nombre d'opérations sur les sytèmes de fichiers qu'elles requièrent. Les commandes impliquant de nombreuses consultations de fichiers, telles que **find**, ou de lectures/écritures, telles que **cp**, sont exécutées beaucoup plus lentement sur un système sans disque. La figure "Résultats des tests" donne une idée des performances sur un système sans disque de quelques commandes usuelles.



Résultats des tests

La pénalité subie par une commande sur un client sans disque est exprimée comme le rapport entre le temps requis sur système à disque et le temps requis sur un système sans disque. Ce taux est intéressant, mais pas toujours important. Par exemple, si une commande demande 0,05 seconde sur un système à disque et 0,2 seconde sur un système sans, la pénalité est de quatre. Mais est-ce vraiment important pour l'utilisateur final ? Un temps de réponse de 0,2 seconde est tout à fait suffisant pour un individu normal... Mais si la commande se trouve dans un script shell et est exécutée 100 fois, le temps de réponse du script passe de 5 à 20 secondes, ce qui est déjà plus gênant. C'est pourquoi il convient d'éviter d'attribuer des stations sans disque à des utilisateurs qui lancent souvent des scripts shell complexes.

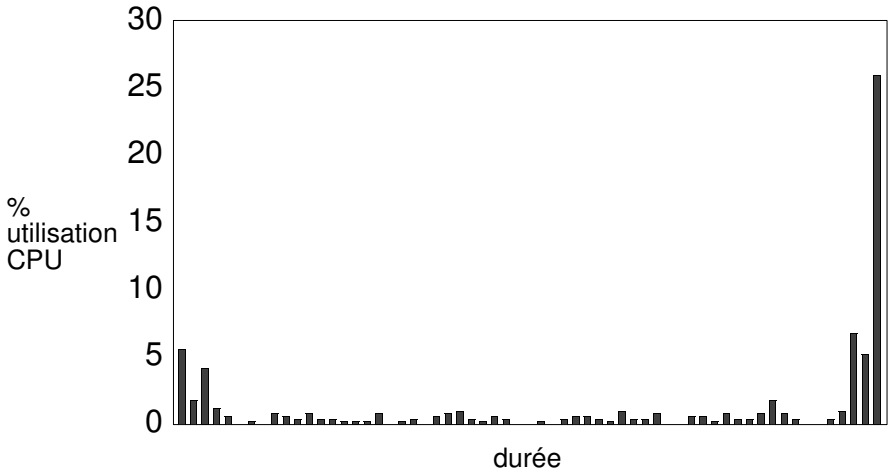
Etude de cas 1 – Environnement de bureau

Pour étudier les caractéristiques des E/S client, nous avons mesuré la charge représentant un environnement de bureau avec un seul utilisateur par client, sur un ESCALA modèle 220 sans disque 16 Mo. La charge étudiée consiste à créer un fichier, à l'aide de l'éditeur **vi**, avec une vitesse de frappe de 6 caractères/seconde, à exécuter les utilitaires **nroff**, **spell** et **cat** sur le document. Le document est transféré via **tftp** sur le serveur. D'autres commandes sont également lancées : **cal**, **calendar**, **rm** et **mail**, ainsi qu'un petit programme qui consulte les numéros de téléphone. Des "temps de réflexion" simulés sont intercalés entre les commandes.

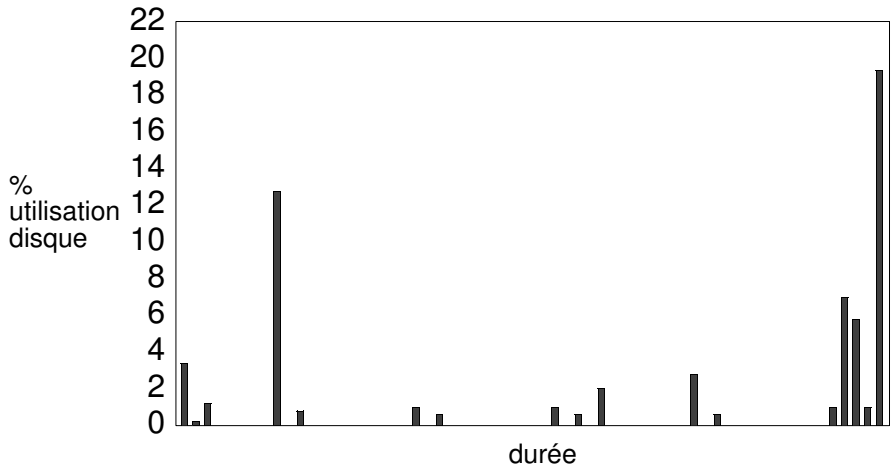
Les figures "Utilisation du CPU serveur dans un environnement bureautique" et "Utilisation du disque serveur dans un environnement bureautique" illustrent l'utilisation des ressources du disque et du CPU du serveur pour la charge étudiée. Le serveur est un modèle 530H avec un disque dur de 857 Mo. Le client exécutant le travail est un seul modèle 220. La charge est intermittente ("en rafales") – les pics d'utilisation étant disproportionnés par rapport au taux moyen.

La figure "Paquets/seconde dans un environnement bureautique Ethernet" illustre les variations du taux de requêtes d'E/S pendant la durée d'exécution du travail. Le taux NFS moyen est de 9.5 requêtes/seconde, avec une pointe à 249 requêtes/seconde. La figure "Octets/seconde dans un environnement bureautique Ethernet" indique le nombre d'octets transférés par seconde (activité du protocole comprise). Le taux de transfert moyen est de 4 000 octets/seconde, avec une pointe à 114 341 octets/seconde. Ce travail consomme en moyenne 1/300ème de la largeur de bande nominale d'un réseau Ethernet, avec une pointe à 1/11ème d'utilisation.

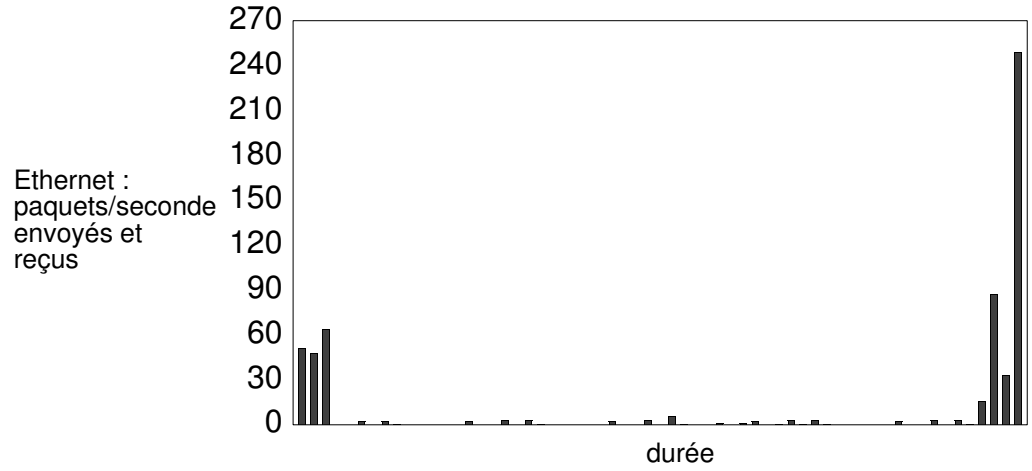
La moyenne d'utilisation de CPU serveur par client étant de 2% %, la moyenne d'utilisation de disque serveur par client de 2.8% % et la moyenne d'utilisation du réseau Ethernet de 0.3% %, le disque sera probablement la ressource critique si plusieurs occurrences de ce travail font appel à un seul serveur.



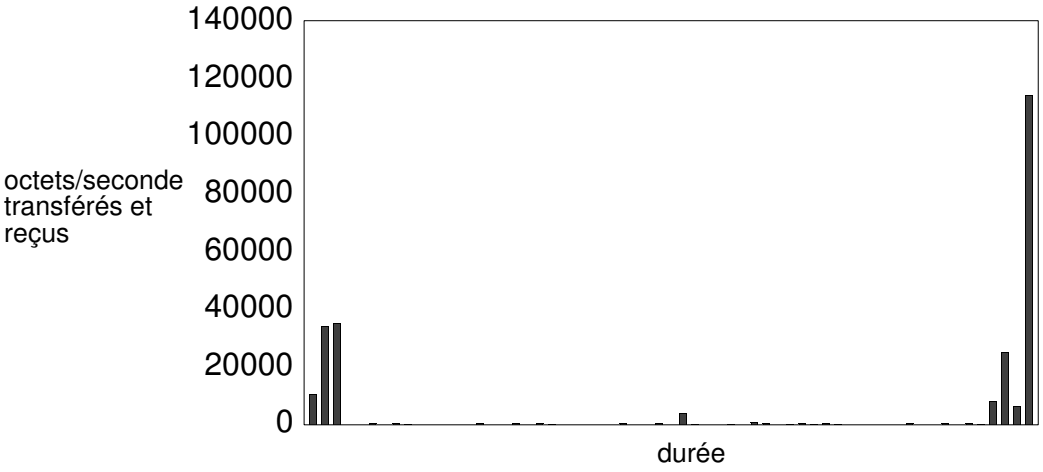
Utilisation de la CPU serveur dans un environnement bureautique



Utilisation du disque serveur dans un environnement bureautique



Paquets/seconde dans un environnement bureautique Ethernet

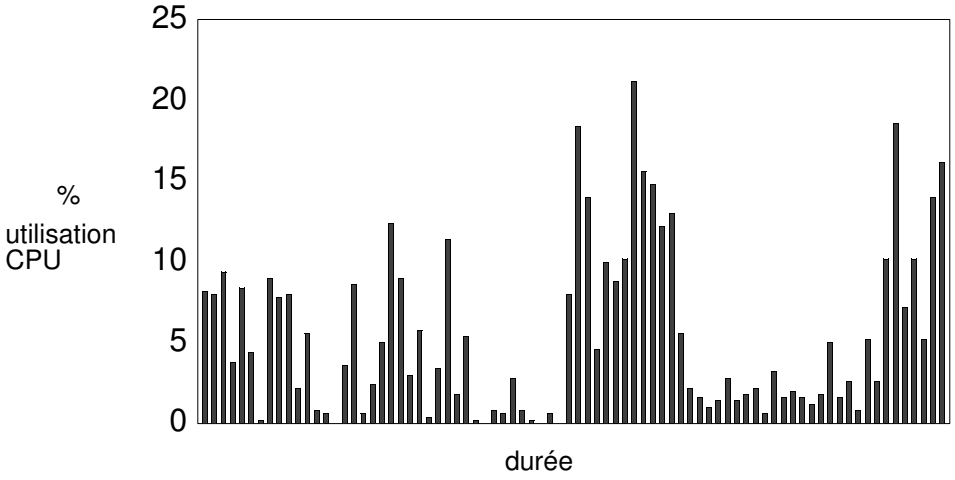


Octets/seconde dans un environnement bureautique Ethernet

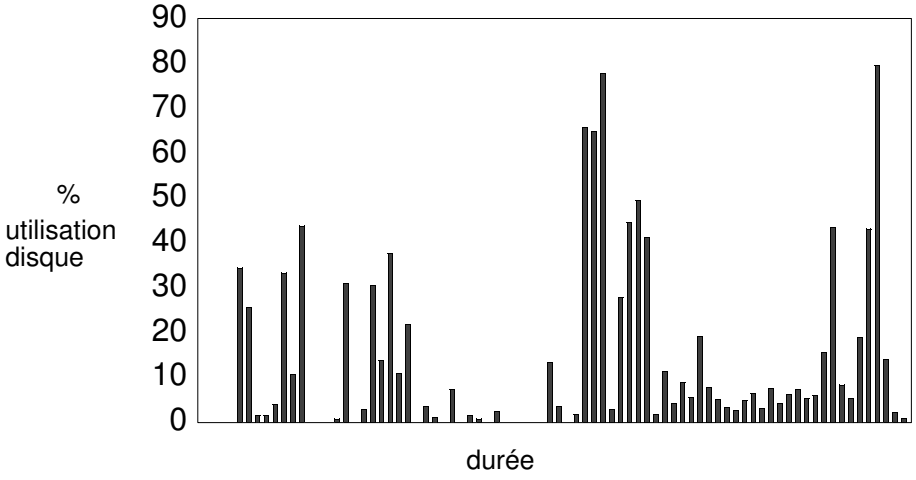
Etude de cas 2 – Environnement de développement de logiciels

Autre exemple de caractéristiques d'E/S client, nous avons mesuré une charge de compilation/établissement de liens/exécution sur un ESCALA modèle 220 16 Mo sans disque. Il s'agit d'une charge autrement importante que celle de l'exemple précédent. Ce travail combine plusieurs services AIX couramment utilisés dans le cadre du développement de logiciels, dans un environnement utilisateur unique par client. Des "temps de réflexion" simulés sont intercalés pour tenir compte du délai de frappe.

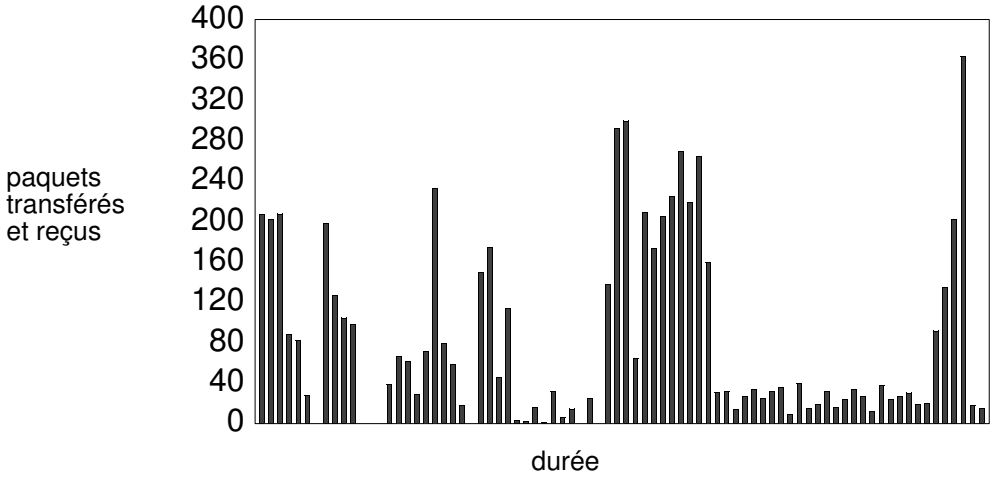
Les figures "Développement de logiciels : utilisation de la CPU serveur" et "Développement de logiciels : utilisation du disque serveur" illustrent l'utilisation des ressources du serveur pour cet environnement de travail (avec la même configuration que dans l'exemple précédent).



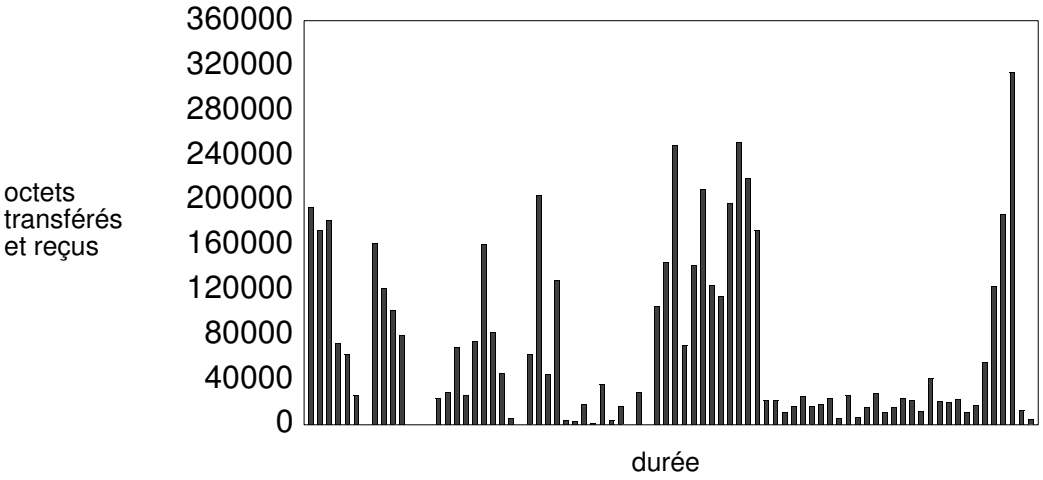
Développement de logiciels : utilisation de la CPU serveur



Développement de logiciels : utilisation du disque serveur



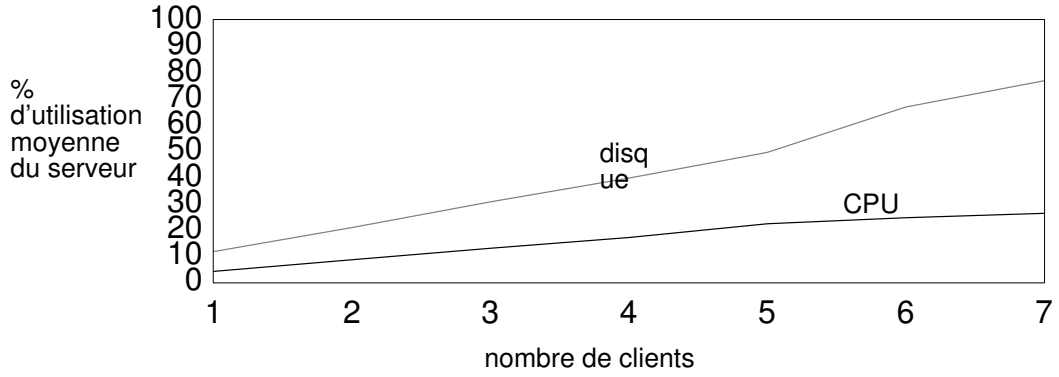
Développement de logiciel : paquets/seconde (Ethernet)



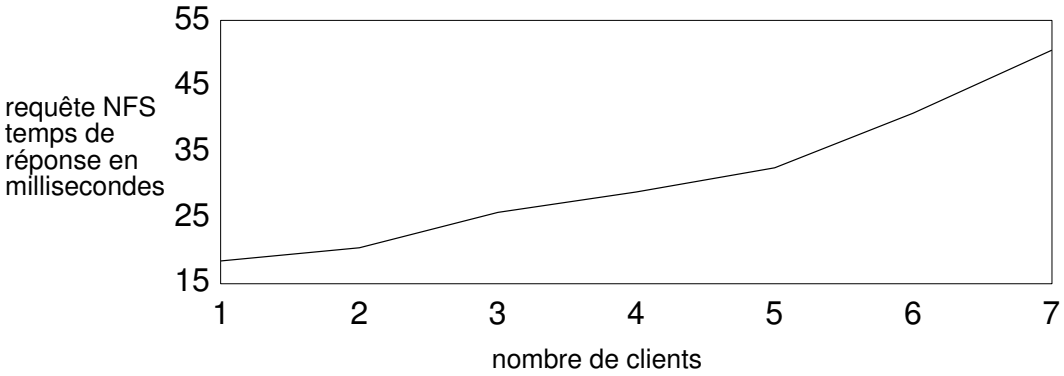
Développement de logiciel : octets/seconde (Ethernet)

La figure ”Paquets/seconde dans un environnement de développement de logiciel Ethernet” illustre les variations du taux de requêtes d’E/S pendant la durée d’exécution du travail. Le taux NFS moyen est de 82 requêtes/seconde, avec une pointe à 364 requêtes/seconde. La figure ”Développement de logiciels : octets/seconde (Ethernet)” illustre le nombre d’octets transférés par seconde (activité du protocole comprise). Le taux de transfert moyen est de 67 540 octets/seconde, avec une pointe à 314 750 octets/seconde. Ce travail consomme en moyenne 1/18ème de la largeur de bande nominale d’un réseau Ethernet, avec une pointe à 1/4ème d’utilisation.

La moyenne d’utilisation de CPU serveur par client étant de 4.2 %, la moyenne d’utilisation de disque serveur par client de 8.9 % et la moyenne d’utilisation du réseau Ethernet de 5.3 %, le disque sera probablement la ressource critique si plusieurs occurrences de ce travail font appel à un seul serveur. Si un second disque a été ajouté à la configuration du serveur, c’est le réseau Ethernet qui saturera sans doute. Il y a toujours un ”goulot d’étranglement suivant”.



Utilisation moyenne du serveur



Temps de réponse aux requêtes NFS

En supposant que le goulot d'étranglement disque s'est produit avec un nombre réduit de clients pour cette charge de travail, il est facilement mesurable. La figure "Utilisation moyenne du serveur" illustre l'utilisation moyenne de CPU et du disque en fonction du nombre de clients (serveur d'un disque). La figure "Temps de réponse aux requêtes NFS" illustre l'évolution du temps de réponse en fonction du nombre de clients.

Voir aussi

"Optimisation de NFS"

Commandes **iostat**, **nfsstat**, **rm** et **smit**.

Démons **biod** and **nfsd**.

Optimisation des connexions asynchrones pour transferts haut débit

Les ports asynchrones permettent de raccorder à un ordinateur des périphériques en option, tels que terminaux, imprimantes, télécopieurs et modems. Ces ports se trouvent sur des unités telles que les cartes BULL 8, 16 ou 64 ports, ou la carte Digiboard 128 ports, qui résident sur le Micro Channel et fournissent de nombreuses connexions asynchrones, généralement RS-232 ou RS-422. Nombre de cartes, comme les cartes asynchrones BULL précitées, furent conçues à l'origine pour servir terminaux et imprimantes, et sont donc optimisées pour les sorties (envois). Le traitement des entrées (réception) est moins optimisé, peut-être parce qu'il a été décrété qu'un utilisateur ne pourrait jamais taper à une vitesse telle qu'il faille se préoccuper de l'optimisation des entrées clavier. Le problème ne se pose pas non plus lorsque la transmission des données est lente et irrégulière, comme les entrées au clavier. Mais il se pose avec les applications en mode brut, où des blocs importants d'entrée sont transmis par d'autres ordinateurs ou des machines telles que les photocopieurs.

Cette section traite des performances respectives des différentes cartes lors des transferts (envoi et réception) de fichiers en mode brut. Nous indiquerons les techniques et les méthodes permettant d'obtenir les meilleures performances lors de ces transferts, en dépit des limitations inhérentes à certaines cartes.

Configurations et objectifs des mesures

Nos mesures ont un double objectif : évaluer le débit en sortie, le débit effectif en bauds et le taux d'utilisation de CPU à différents débits pour les cartes, et déterminer le nombre maximal de ports susceptibles d'être pris en charge par chaque unité en fonction des différents débits.

> **Remarque :** Les mesures de débit sortant, effectuées sur des travaux dont les transferts de fichiers sont exécutés en mode brut, sont particulièrement utiles pour estimer les performances d'unités fonctionnant en mode brut, tels les télécopieurs et les modems. Ces mesures ne s'appliquent pas aux configurations multi-utilisateurs commerciales, dont la CPU peut être fortement sollicitée par les accès aux bases de données ou la gestion de l'écran, et qui sont généralement limitées au niveau des E/S disque.

En mode brut, les données sont traitées comme un flux continu : les octets en entrées ne sont pas regroupés en lignes et les fonctions "supprimer" et "tuer" sont désactivées. Une taille minimale de bloc de données et une horloge de lecture permettent de déterminer le traitement appliqué aux octets par le système, avant transmission à l'application.

Les mesures ont été effectuées sur les cartes natives 8, 16 et 64 ports, aux vitesses de 2 400, 9 600, 19 200 et 38 400 bauds. (Les ports asynchrones natifs et les cartes 8 et 16 ports du ESCALA étant tous servis par le même pilote et leurs performances étant similaires, ils sont référencés sous le même terme, la carte 8/16 ports.) La carte 128 ports n'a été testée qu'à 19 200 et 38 400 bauds.

La commande AIX de contrôle des performances, **iostat** (ou **sar**, pour la carte 128 ports), a été exécutée en arrière-plan à une fréquence prédéfinie, pour tester les performances du système. Des mesures ont été effectuées dès qu'un état restait stable pendant un intervalle de temps fixe, dans la portion rectiligne de la courbe de sortie. Pour chaque test, la charge système a été progressivement accrue par l'ajout de ports actifs – jusqu'au maximum admis ou l'utilisation de la CPU à 100 %).

Trois mesures significatives au niveau des performances de pointe – sortie moyenne cumulée de caractères par seconde (car/sec) sur l'intervalle considéré, débit effectif par ligne et utilisation de la CPU – ont été effectuées pour la réception et l'envoi semi-duplex.

La régulation XON/XOFF (établissement de connexion asynchrone, sans relation avec la régulation d'E/S disque AIX), la régulation RTS/CTS et l'absence de régulation ont été testées. Régulation et établissement de connexion font référence à des mécanismes

logiciels ou matériels, exploités en communication, pour interrompre les transmissions lorsque l'unité réceptrice ne peut plus stocker les données qu'elle reçoit. Nous avons constaté l'adéquation de la régulation XON/XOFF avec les cartes 8/16 ports pour la réception, et avec la carte 128 ports pour l'envoi et la réception. RTS/CTS est plus adapté à la carte 64 ports en réception. Pour l'envoi, l'absence de régulation s'est imposée pour les cartes 8/16 et 64 ports.

La sortie de caractères est la somme cumulée des caractères transmis par seconde sur toutes les lignes. Les vitesses des lignes (débit en bauds) de 2 400, 9 600, 19 200 et 38 400, définies par le logiciel, sont optimales pour le transfert des données via des lignes TTY. Alors que le débit correspond à la vitesse de pointe de la ligne, exprimée en bits/seconde, le débit *effectif,* toujours moindre, est calculé comme étant égal à 10 fois le débit de caractère divisé par le nombre de lignes. (Le produit par 10 est justifié par le fait que le tranfert d'un caractère de 8 bits utilise 10 bits.)

Résultats

Le tableau suivant récapitule les résultats obtenus. "Max ports:" est le nombre maximal de ports pris en charge par la carte, lorsque le débit effectif est proche de la vitesse de la ligne.

Vitesse ligne	**8/16 ports :**		**64 ports :**		**128 ports :**	
	Send	Receive	Send	Receive	Send	Receive
2400 baud						
Max ports:	32	16	64	64	N/A	N/A
Char/sec	7700	3800	15200	14720		
Eff. Kb/sec:	2.4	2.4	2.3	2.3		
CPU util. %:	5	32	9	76		
9600 baud						
Max ports:	32	12	56	20	128	128
Char/sec	30700	11500	53200	19200	122200	122700
Eff. Kb/sec:	9.6	9.6	9.5	9.6	9,6	9.6
CPU util. %:	17	96	25	99	21	27
19,200 baud						
Max ports:	32	6	32	10	128	128
Char/sec	48900	11090	51200	18000	245400	245900
Eff. Kb/sec:	15.3	18.5	16	18	19.2	19.2
CPU util. %:	35	93	23	92	39	39
38400 baud						
Max ports:	32	4	24	7	75	75
Char/sec	78400	10550	50400	15750	255200	255600
Eff. Kb/sec:	24.5	26.4	21	22.5	34	34
CPU util. %:	68	98	23	81	40	37

Carte asynchrone 8/16 ports

Envoi semi-duplex 8/16 ports

Les mesures relatives aux envois semi-duplex 8/16 ont été effectuées sans régulation, permettant la libre transmission des données. Pour la carte 8/16 ports, le ESCALA traite environ 1 400 car/sec pour 1 % d'utilisation CPU. La pointe de sortie d'une seule carte 16 ports est de 48 000 car/sec.

Réception semi-duplex 8/16 ports

Dans cette configuration, avec régulation XON/XOFF, le ESCALA traite environ 120 car/sec pour 1 % d'utilisation CPU. La pointe de largeur de bande est de 11 000 car/sec à 100 % d'utilisation CPU pour la carte asynchrone 16 ports.

Carte asynchrone 64 ports

Dans les systèmes à carte asynchrone 64 ports, les limites proviennent le plus souvent du boîtier concentrateur 16 ports (il peut y en avoir quatre). La saturation du concentrateur est un problème car, lorsqu'elle approche, aucun transit n'est plus accepté. Le débit effectif est

réduit, et on constate une importante baisse d'activité. Pour les mesures qui suivent, quatre concentrateurs 16 ports ont été connectés aux 64 ports RS-232.

Réception semi-duplex 64 ports

Les mesures relatives à la réception semi-duplex 64 ports ont été effectuées avec régulation matérielle RTS/CTS. Dans cette configuration, le ESCALA traite environ 195 car/sec pour 1 % d'utilisation CPU. La pointe de largeur de bande est de 19 500 car/sec à 100 % d'utilisation CPU.

En réception semi-duplex, un boîtier concentrateur unique 16 ports sature à 8 450 car/sec avec 44 % d'utilisation CPU. Une fois le concentrateur saturé, aucun transit n'est possible – tant qu'un autre concentrateur n'est pas installé. A 38 400 bauds, le point de saturation d'un concentrateur unique est atteint avec quatre ports actifs et un débit effectif de 22,5 kbauds. A 19 200 bauds, le point de saturation est atteint avec cinq ports actifs et un débit effectif de 17 kbauds. A 9 600 bauds, le point de saturation est atteint avec neuf ports actifs et un débit effectif de 9,6 kbauds. A 2 400 bauds, le système accepte les 64 ports avec un débit effectif de 2,3 kbauds sans point de saturation. La pointe de débit est de 14 800 car/sec.

Envoi semi-duplex 64 ports

Les mesures relatives aux envois semi-duplex 64 ports ont été effectuées sans régulation, permettant la libre transmission des données, sans restriction au niveau du contrôle de flux. Pour la carte 64 ports, le ESCALA traite environ 1 400 car/sec pour 1 % d'utilisation CPU. La pointe de débit de la carte 64 ports avec les quatre concentrateurs est de 54 500 car/sec.

Un boîtier concentrateur unique sature à 13 300 car/sec avec 6 % d'utilisation CPU. A 38 400 bauds, il accepte six ports avec un débit effectif d'environ 22 kbauds. A 19 200 bauds, il accepte huit ports avec un débit effectif d'environ 16,3 kbauds.

Carte asynchrone 128 ports

Sept cartes asynchrones Digiboard 128 ports peuvent être connectées à un ESCALA, pour un total de 896 ports.

Il existe deux liaisons SDLC (synchronous-data-link-control) par carte, avec une capacité cumulée de 2,4 Mbauds. (La carte 64 ports est dotée d'un SDLC quatre canaux pour une capacité cumulée de 768 kbauds.)

D'autres caractéristiques de la carte 128 ports améliorent la vitesse de transmission des données et réduisent le taux d'utilisation CPU :

- L'algorithme d'interrogation prend en charge l'interruption d'horloge, évitant toute interruption hôte supplémentaire. Les taux d'interrogation peuvent être modifiés par l'application – port par port.
- Le pilote de l'unité détecte les E/S en mode brut et déplace les données de la mémoire de la carte vers l'espace utilisateur, passant outre la procédure de transmission hôte.
- Le concentrateur traite la plupart des options de discipline de ligne. Le code "préparé" est une exception : tous les traitements sont effectués par l'hôte.
- Le microcode de la carte réaffecte les tampons mémoire en fonction du nombre de concentrateurs et de la mémoire disponible.

Le concentrateur ne sature pas avec les cartes asynchrones 128 ports, donnant à cette carte un avantage certain sur les cartes asynchrones 64 ports.

Pour les mesures, huit boîtiers concentrateurs 16 ports ont été connectés aux 128 ports RS-232.

Réception semi-duplex 128 ports

Avec la régulation logicielle XON/XOFF, cette configuration traite environ 6 908 car/sec pour 1% d'utilisation CPU. La pointe de débit est de 255 600 car/sec à 37 % d'utilisation CPU.

Envoi semi-duplex 128 ports

Sans régulation, le taux maximal auquel cette configuration peut envoyer des données à une unité TTY est d'environ 5 800 car/sec pour 1 % d'utilisation CPU. La pointe de sortie de la carte 128 ports est 255 200 car/sec.

Techniques d'optimisation du port asynchrone

Les configurations test de cette étude font appel à nombre de mécanismes logiques et physiques de contrôle de flux, ainsi qu'à des techniques logicielles pour optimiser les taux de transmission de caractères. Vous trouverez ci-dessous quelques précisions sur ces techniques. (Un script shell contenant les commandes **stty** permettant d'implanter l'essentiel de ces techniques est fourni en fin de section.)

- Augmentez la valeur (4, par défaut) de la variable *vmin* pour chaque TTY. *vmin* est le nombre minimal d'octets à recevoir lorsque la lecture aboutit. La valeur de *vmin* doit être la plus petite de la taille de bloc des données ou 255 (maximum autorisé). Si la taille de bloc de l'application est variable ou inconnue, définissez *vmin* à 255 : le nombre de lectures sera réduit de même que le taux d'utilisation de CPU (de 15 à 20 % pour les programmes de transfert de fichiers).
- Sauf sur la carte 128 ports, définissez *vtime* > 0 pour prévenir une lecture de bloc indéfinie. Si *vtime* est défini à zéro sur la carte 128 ports, le traitement de la procédure de transmission POSIX sera déchargé sur le matériel de la carte, réduisant sensiblement le traitement CPU.
- Pour les envois en mode brut où aucune translation en sortie n'est nécessaire, désactivez l'option **opost** sur ligne de procédure POSIX. Les performances CPU seront améliorées par la réduction de la longueur du chemin de sortie. Pour les applications de transfert de fichiers, qui déplacent d'importantes quantités de données sur les lignes TTY, le taux d'utilisation de CPU peut être divisé par 3. Exemple :

  ```
  # stty -opost < /dev/ttyn
  ```

- La carte 64 ports étant sujet à des surcharges de données imprévisibles lorsque le débit est élevé et que la régulation est effectuée via XON/XOFF, optez pour la régulation matérielle RTS/CTS. Vous évitez ainsi les risques de perte de données.
- Dans la mesure où les boîtiers concentrateurs de la carte 64 ports ont une largeur de bande limitée et saturent lorsque les débits sont élevés, ajouter des ports à un concentrateur saturé ne fera que dégrader les performances de tous les ports connectés. Ajoutez plutôt un autre concentrateur et poursuivez jusqu'à ce que lui aussi sature ou que la CPU soit surchargée.
- Pour le traitement des entrées, l'option **echo** est coûteuse, dans la mesure où elle accroît la durée de traitement par caractère. L'écho de caractère est utile pour les entrées utilisateur canoniques, mais le plus souvent inutile pour la plupart des applications en mode brut. Exemple :

  ```
  # stty -echo < /dev/ttyn
  ```

Transfert rapide de fichiers avec fastport

Le script `fastport.s` a pour objet de configurer un port TTY pour le transfert rapide des fichiers en mode brut (lorsqu'un télécopieur doit être raccordé, par exemple). Utiliser ce script peut multiplier par 3 les performances CPU (à un débit de 38 400 bauds). `fastport.s` n'est pas destiné au traitement canonique utilisé lors de l'interaction avec un utilisateur sur terminal asynchrone, car un traitement canonique ne peut pas être aisément placé en mémoire tampon. La largeur de bande d'une lecture canonique est trop faible pour les paramètres du port rapide pour qu'une différence soit perceptible.

N'importe quel port TTY peut être configuré comme port rapide. L'amélioration des performances qui en résulte est due à la diminution du nombre d'interruptions vers la CPU au cours du cycle de lecture sur une ligne TTY donnée.

1. Créez un TTY pour le port via SMIT (**Unités** –> **TTY** –> **Ajout d'un TTY**), avec Activation de la connexion=disable et Vitesse de transmission=38 400.
2. Créez le script shell Korn `fastport.s`, comme suit :

```
#*************************************************************
#
#            Configures a fastport for ”raw” async I/O.
#
#*************************************************************
set -x
sync;sync
i=$1

if [ $i -le 100 ]
then
# for the native async ports and the 8-, 16-, and 64-port adapters
# set vmin=255 and vtime=0.5 secs with the following stty
 stty -g </dev/tty$i |awk ’ BEGIN { FS=”:”;OFS=”:” }
  { $5=”ff”;$6=5;print $0 } ’ >foo
# for a 128-port adapter, remove the preceding stty, then
# uncomment and use the
# following stty instead to
# set vmin=255 and vtime=0 to offload line discipline processing
# stty -g </dev/tty$i |awk ’ BEGIN { FS=”:”;OFS=”:” }
#  { $5=”ff”;$6=0;print $0 } ’ >foo
 stty ‘cat foo ‘ </dev/tty$i
 sleep 2

# set raw mode with minimal input and output processing
 stty -opost -icanon -isig -icrnl -echo -onlcr</dev/tty$i
   rm foo
 sync;sync
else
   echo ”Usage is fastport.s < TTY number >”
fi
```

3. Appelez le script pour le *numéro* TTY avec la commande :

```
fastport.s number
```

Evaluation des performances réseau avec netpmon

La commande **netpmon** fait appel à la fonction de suivi pour obtenir une image détaillée de l'activité du réseau pendant un intervalle de temps déterminé. Dans la mesure où la commande **filemon** invoque cet utilitaire, seul l'utilisateur `root` ou un membre du groupe `system` est habilité à la lancer. La commande **netpmon** n'est pas prévue pour fonctionner avec NFS3(ONC+).

Sous AIX version 4.1, la commande **netpmon** est intégrée à la boîte à outils PTX (Performance Toolbox for AIX). Pour déterminer si **netpmon** est disponible, entrez :

```
lslpp -lI perfagent.tools
```

Si ce module est installé, **netpmon** est disponible.

Le suivi, lancé par la commande **netpmon**, peut être suspendu par **trcoff,** relancé par **trcon** et arrêté par **trcstop**. Dès que le suivi est terminé, **netpmon** en envoie le compte rendu à **stdout**. Voici un exemple d'utilisation de **netpmon** :

```
# netpmon -o nm.test.out ; ping xactive 256 5 ; trcstop
```

Le compte rendu généré (quelque peu résumé) par cette séquence, dans un système au repos, est le suivant :

```
Wed Jan 12 14:33:25 1994
System: AIX alborz Node: 3 Machine: 000249573100
4.155 secs in measured interval
========================================================================
Process CPU Usage Statistics:
-----------------------------
                                                     Network
Process (top 20)             PID  CPU Time   CPU %   CPU %
-------------------------------------------------------------
ping                       12699    0.0573   1.380   0.033
trcstop                    12700    0.0150   0.360   0.000
ksh                        13457    0.0150   0.360   0.000
rlogind                     6321    0.0127   0.306   0.088
netpmon                    12690    0.0064   0.153   0.000
netw                         771    0.0047   0.113   0.113
netpmon                    10650    0.0037   0.090   0.000
trace                      10643    0.0023   0.055   0.000
swapper                        0    0.0022   0.053   0.000
writesrv                    1632    0.0009   0.021   0.000
-------------------------------------------------------------
Total (all processes)               0.1201   2.891   0.234
Idle time                           3.8904  93.639
========================================================================
First Level Interrupt Handler CPU Usage Statistics:
---------------------------------------------------
                                                     Network
FLIH                               CPU Time   CPU %   CPU %
-------------------------------------------------------------
external device                      0.0573   1.379   0.890
data page fault                      0.0368   0.887   0.000
floating point                       0.0001   0.003   0.000
-------------------------------------------------------------
Total (all FLIHs)                    0.0943   2.269   0.890
========================================================================
Second Level Interrupt Handler CPU Usage Statistics:
----------------------------------------------------
                                                     Network
SLIH                               CPU Time   CPU %   CPU %
-------------------------------------------------------------
clock                                0.0415   0.998   0.000
tokdd                                0.0064   0.154   0.154
<addr=0x00022140>                    0.0008   0.019   0.000
-------------------------------------------------------------
Total (all SLIHs)                    0.0486   1.171   0.154
```

```
================================================================================
Network Device-Driver Statistics (by Device):
------------------------------------------
                 ----------- Xmit ----------   ----- Recv ----
Device           Pkts/s  Bytes/s  Util  QLen   Pkts/s  Bytes/s
------------------------------------------------------------
/dev/tok0          3.37      629 0.005 0.005    16.85     1900
================================================================================
Network Device-Driver Transmit Statistics (by Destination Host):
----------------------------------------------------------------
Host                     Pkts/s  Bytes/s
---------------------------------------
xactive.austin.ibm.com     1.44      390
================================================================================
Detailed Second Level Interrupt Handler CPU Usage Statistics:
-------------------------------------------------------------
SLIH: tokdd
count:                  84
  cpu time (msec):      avg 0.076   min 0.058   max 0.097   sdev 0.009
================================================================================
Detailed Network Device-Driver Statistics:
------------------------------------------
DEVICE: /dev/tok0
recv packets:           70
  recv sizes (bytes):   avg     112,8 min       68 max      324 sdev
75,2
  recv times (msec):    avg     0,226 min       0,158 max      0,449 sdev
   0,056
xmit packets:           14
  xmit sizes (bytes):   avg     186,6 min       52 max      314 sdev
100,0
  xmit times (msec):    avg     1,552 min       1,127 max      2,532 sdev
   0,380
================================================================================
Detailed Network Device-Driver Transmit Statistics (by Host):
-------------------------------------------------------------
HOST: xactive.austin.ibm.com
xmit packets:           6
  xmit sizes (bytes):   avg     270,3 min       52 max      314 sdev
97,6
  xmit times (msec):    avg     1,772 min       1,516 max      2,532 sdev
   0,346
```

Analyse des problèmes de performance avec iptrace

Il existe de nombreux outils de suivi de l'activité, normale ou anormale, sur le réseau. Certains sont opérationnels sous AIX, d'autres requièrent un matériel spécifique. Un des outils permettant d'obtenir une description précise, paquet par paquet, de l'activité du LAN générée par une charge de travail, est la combinaison du démon **iptrace** et de la commande **ipreport**. Seul l'utilisateur `root` est habilité à lancer le démon **iptrace**.

Par défaut, **iptrace** assure le suivi de tous les paquets. Une option (**–a**) permet d'exclure les paquets du protocole ARP (Address Resolution Protocol). D'autres options permettent de resteindre l'objet du suivi à un hôte source (**–s**), un hôte destinataire (**–d**) ou un protocole (**–p**) donné. Reportez-vous au manuel *AIX Commands Reference*. La consommation **iptrace** de temps processeur étant non négligeable, ciblez bien les paquets dont vous souhaitez le suivi.

iptrace étant un démon, mieux vaut le lancer via une commande **startsrc** et non à partir de la ligne de commande : il sera plus facile de le contrôler et de l'arrêter "proprement". Voici un exemple d'appel classique :

```
# startsrc –s iptrace –a "–i tr0 /home/user/iptrace/log1"
```

Cette commande lance le démon **iptrace** avec mission d'assurer le suivi de toutes les activités sur l'interface en anneau à jeton, `tr0`, et de placer les données de suivi dans `/home/user/iptrace/log1`. Pour arrêter le démon, entrez :

```
# stopsrc –s iptrace
```

Si vous ne l'aviez pas lancé avec **startsrc**, vous auriez dû rechercher son ID processus via **ps** et le tuer (**kill**).

La commande **ipreport** est une commande de formatage pour le fichier journal. Sa sortie est écrite sur `stdout`. Des options permettent de reconnaître et de formater les paquets RPC (**–r**), d'identifier chaque paquet par un numéro (**–n**) et de préfixer chaque ligne par une chaîne de 3 caractères identifiant le protocole (**–s**). Voici une commande **ipreport** formatant le fichier `log1` juste créé (appartenant à l'utilisateur `root`) :

```
# ipreport –ns log1 >log1–formaté
```

Il en résulte une séquence de comptes rendus de paquet semblable aux exemples ci-après. Le premier paquet est la première moitié d'un **ping**. Les champs les plus intéressants sont : les adresses des hôtes source (SRC) et destination (DST) (en décimal avec point séparateur et en ASCII), la longueur du paquet IP (ip_len), et l'indication du protocole de plus haut niveau en cours (ip_p).

```
Packet Number 131
TOK: =====( packet transmitted on interface tr0 )=====Fri Dec 10 08:42:07 1993
TOK : 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 90:00:5a:a8:88:81, dst = 10:00:5a:4f:35:82]
TOK: routing control field = 0830,  3 routing segments
TOK: routing segments [ ef31 ce61 ba30  ]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =  129.35.145.140 >  (alborz.austin.ibm.com)
IP:     < DST =  129.35.145.135 >  (xactive.austin.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=38892, ip_off=0
IP:     ip_ttl=255, ip_sum=fe61, ip_p = 1 (ICMP)
ICMP:   icmp_type=8 (ECHO_REQUEST)  icmp_id=5923  icmp_seq=0
ICMP: 00000000     2d088abf 00054599 08090a0b 0c0d0e0f     |-.....E.........|
ICMP: 00000010     10111213 14151617 18191a1b 1c1d1e1f     |................|
ICMP: 00000020     20212223 24252627 28292a2b 2c2d2e2f     | !"#$%&'()*+,-./|
ICMP: 00000030     30313233 34353637                       |01234567        |
```

L'exemple suivant est une trame résultant d'une opération **ftp**. Notez que le paquet IP est la taille du MTU pour ce réseau (1 492 octets).

```
Packet Number 501
TOK: =====( packet received on interface tr0 )=====Fri Dec 10 08:42:51 1993
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 18, frame control field = 40
TOK: [ src = 90:00:5a:4f:35:82, dst = 10:00:5a:a8:88:81]
TOK: routing control field = 08b0,  3 routing segments
TOK: routing segments [ ef31 ce61 ba30  ]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:     < SRC =  129.35.145.135 >  (xactive.austin.ibm.com)
IP:     < DST =  129.35.145.140 >  (alborz.austin.ibm.com)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=1492, ip_id=34233, ip_off=0
IP:     ip_ttl=60, ip_sum=5ac, ip_p = 6 (TCP)
TCP:    <source port=20(ftp-data), destination port=1032 >
TCP:    th_seq=445e4e02, th_ack=ed8aae02
TCP:    th_off=5, flags<ACK |>
TCP:    th_win=15972, th_sum=0, th_urp=0
TCP: 00000000     01df0007 2cd6c07c 00004635 000002c2     |....,..|..F5....|
TCP: 00000010     00481002 010b0001 000021b4 00000d60     |.H........!....`|
             --------- Lots of uninteresting data omitted -----------
TCP: 00000590     63e40000 3860000f 4800177d 80410014     |c...8`..H..}.A..|
TCP: 000005a0     82220008 30610038 30910020              |.”..0a.80..     |
```

Chapitre 10. Optimisation des performances DFS

Remarque : Les recommandations qui suivent s'appuient sur des essais effectués sous AIX version 4.1. Au moment où nous écrivons, nous ne savons pas dans quelle mesure elles s'appliquent également à AIX version 4.1.

Du point de vue des performances, la différence essentielle entre DCE DFS et NFS est la capacité du client de mise en cache de DFS. Il n'est donc pas étonnant que les principales techniques d'optimisation de DFS concernent les attributs du cache client. Les paramètres client et serveur décrits dans ce chapitre concernent :

- "Cache DFS sur disque ou en mémoire"
- "Taille du cache DFS"
- "Taille de la tranche de cache DFS"
- "Nombre de tranches de cache DFS"
- "Emplacement du cache disque DFS"
- "Taille du tampon d'état du cache DFS"
- "Taille des articles et lectures/écritures"
- "Paramètres de communication DFS"
- "Optimisation du serveur de fichiers DFS"
- "Optimisation LFS DCE pour les performances DFS."

Cache DFS sur disque ou en mémoire

Pour déterminer si, dans votre environnement, mieux vaut accéder au disque ou à la mémoire, considérez ce qui suit :

- Si le système en cours d'optimisation, ou un autre système présentant une charge de travail similaire, exploite déjà DFS avec un cache disque, vous pouvez estimer la taille de cache mémoire requise, en lançant, à l'issue d'une période de pointe, la commande :

  ```
  cm getcachesize
  ```

  Divisez le nombre de blocs de 1 ko utilisés par 0,9 pour déterminer la taille de cache mémoire requise pour la même quantité de données. (Environ 10 % des blocs du cache servent à conserver les articles DFS.)

- Si l'accès à ces données est fréquent, optez pour un potentiel supérieur de cache disque.
- Si la quantité de données gérées est telle que le cache disque le plus grand risque d'être submergé, ou que les données sont fréquemment modifiées par un autre client, optez de préférence pour un cache mémoire, plus efficace au niveau des performances RFC.
- La taille d'un cache mémoire ne doit pas excéder 10 % de la taille de la mémoire réelle de la machine. La taille recommandée est d'environ 5 % de la mémoire réelle. DFS exploitant la capacité de mise en cache mémoire du VMM AIX, l'essentiel du cache mémoire DFS est occupé par les informations sur les répertoires et les points de montage.
- Si vous constatez un signe de limitation en mémoire du système (valeur non nulle à la colonne `pi` ou `po` d'un relevé **vmstat**), n'utilisez pas de cache mémoire pour DFS.

- A titre de test de faisabilité, vous pouvez réduire temporairement, via **rmss**, la taille effective de la mémoire en déduisant la quantité de mémoire que vous affecterez au cache. Si vous constatez une activité de pagination ou une baisse de performances, ou les deux, abandonnez l'idée de créer un cache mémoire. Reportez-vous à "Estimation de la mémoire requise via **rmss**", page 7-6.

Taille du cache DFS

La détermination de la taille adéquate du cache DFS pour un système donné se fait un peu par tâtonnement. Vous pouvez commencer par estimer la somme :

- des tailles de l'ensemble des fichiers de données DFS résidants lus au moins une fois par jour,
- des quantités de données DFS résidantes générées quotidiennement par les utilisateurs,
- des tailles des programmes DFS résidants exécutés plus d'une fois par jour.

Si les répertoires personnels des utilisateurs se trouvent dans DFS, vous pouvez définir des autorisations concernant la fréquence d'accès à ces répertoires, et étudier la réaction du système.

La taille du cache client, spécifiée dans le fichier **CacheInfo**, peut être remplacée via l'option **dfsd –blocks** *n*, *n* étant le nombre de ko du cache. Ce paramètre s'applique aux caches mémoire comme aux caches disque.

Taille de la tranche de mémoire cache DFS

La taille de tranche du cache DFS varie entre 8 et 256 ko. Pour les fichiers volumineux (plusieurs Mo), les performances des lectures/écritures séquentielles augmentent avec cette taille (jusqu'à environ 64 ko). Pour les fichiers encore plus importants (plus de 100 Mo), une tranche de 256 ko entraîne la meilleure performance en lecture.

La taille de tranche est spécifiée via l'option **dfsd –chunksize** *n*, *n* étant un entier compris entre 13 et 18 (inclus). La taille du cache est de 2***n* octets, et est dons comprise entre 8 (2**13) et 256 ko (2**18). Ce paramètre s'applique aux caches mémoire comme aux caches disque. Taille par défaut : ko (caches mémoire) et 64 ko (caches disque).

Nombre de tranches de cache DFS

Ce paramètre ne s'applique qu'aux caches disque. Pour les caches mémoire, le nombre de tranches est déjà spécifié par la combinaison de la taille de cache et de la taille de tranche. Pour les caches disque, le nombre de tranches par défaut est défini comme étant le nombre de blocs cache divisé par 8. Si un **du** du répertoire cache indique que l'espace n'est rempli qu'à moins de 90 %, augmentez le nombre de tranches de cache via l'option **dfsd –files** *n*, *n* étant le nombre de tranches à adapter. Ce qui permet une meilleure utilisation de l'espace cache disponible pour les applications qui font appel à de nombreux petits fichiers. Plusieurs fichiers ne pouvant partager une tranche, le nombre de tranches détermine le nombre de fichiers susceptibles d'être pris en charge par le cache.

Emplacement du cache disque DFS

Le cache disque doit se trouver dans un volume logique qui :

- se trouve dans la zone `outer_edge`, s'il se trouve sur une unité de disque de 200 Mo, 540 Mo ou 1 Go,
- se trouve dans la zone `center`, sur une unité de disque autre que celles précitées,
- se trouve ailleurs que dans le groupe de volume `rootvg`,

- se trouve sur un autre volume physique que l'espace de pagination,
- soit principalement ou exclusivement occupé par le cache disque,
- soit suffisamment vaste pour contenir le cache disque spécifié sans empiéter sur le reste.

Taille du tampon d'état du cache DFS

La taille du tampon d'état limite le nombre maximal de fichiers susceptibles de se trouver simultanément dans le cache. Une entrée est requise pour chaque fichier. Si le tampon d'état est saturé, les nouveaux fichiers déplaceront les anciens dans le cache, même si l'espace disque est suffisant. Si votre charge de travail concerne essentiellement des fichiers dont la taille est inférieure ou égale à celle de la tranche, le tampon d'état doit avoir autant d'entrées qu'il y a de tranches dans le cache.

La taille du tampon d'état est spécifiée via l'option **dfsd –stat** *n*, *n* étant le nombre d'entrées du tampon d'état. Valeur par défaut de *n* :

Taille des articles et lectures/écritures

Les performances relatives aux lectures/écritures séquentielles sont affectées par la taille des articles lus ou écrits par l'application. En général, le débit des lectures croît avec des articles dont la taille augmente jusqu'à 4 ko, puis décroît au-delà. Le débit des écritures croît avec des articles dont la taille augmente jusqu'à 2 ko, puis se stabilise ou décroît légèrement au-delà.

Paramètres de communication DFS

DFS se sert du protocole de communication UDP. Les recommandations pour optimiser les communication DFS pour les systèmes client multiutilisateur et les serveurs sont celles indiquées pour l'optimisation des communications en général (voir "Récapitulatif des paramètres d'optimisation UDP, TCP/IP et mbuf", page 9-31) :

- Donnez à la taille des files d'attente de réception et d'envoi de la carte réseau la valeur 150 (le maximum). Pour ce faire, faites appel à **smit commodev** –> (type carte) –> Adapter –> Change / Show Characteristics of a (type carte) Adapter. Ces paramètres ne peuvent être modifiés en cours d'utilisation de la carte. SMIT permet de préciser que la modification ne doit prendre effet qu'à l'amorçage suivant du système.

  Vous pouvez également définir ces paramètres via **chdev**, si vous commencez par désactiver la carte. Par exemple, pour une carte en anneau à jeton, la séquence de commande est :

  ```
  # ifconfig tr0 detach
  # chdev -l tok0 -a xmt_que_size=150 -a rec_que_size=150
  # ifconfig tr0 hostname up
  ```

  Vous pouvez observer l'effet de la modification via :

  ```
  $ lsattr -E -l tok0
  ```

- Si la commande **netstat –s** génère un nombre non nul dans `udp : n socket buffer overflows`, augmenter la valeur des paramètres **sb_max** et **udp_recvspace** via la commande **no** ne résoud le problème que si une application autre que DFS subit un dépassement. DFS définit ses propres valeurs (176 ko) pour **sb_max** et **udp_recvspace**. Ces valeurs ne sont ni affichées ni modifiées par la commande **no**.

Optimisation du serveur de fichiers DFS

Sur les serveurs haute vitesse, il peut être souhaitable d'augmenter le nombre de **–mainprocs** et de **–tokenprocs** (dans la commande **fxd**), pour garantir l'exploitation de toute la capacité CPU disponible.

- Commencez par spécifier **–mainprocs 10 –tokenprocs 4**.
- Exécutez **vmstat** en période de pointe. Si vous constatez un niveau très élevé d'E/S CPU en attente, tentez d'augmenter encore les valeurs de **–mainprocs** et de **–tokenprocs**.

Optimisation LFS DCE pour les performances DFS

Lorsque vous configurez un agrégat LFS DCE (via la commande **newaggr**) sur un serveur DFS, tenez compte de ce qui suit :

- Si la plupart des fichiers sont volumineux, donnez au paramètre **–blocksize** la valeur maximale autorisée qui soit inférieure à la taille standard de fichier. La valeur de **–blocksize** peut être une puissance de 2 quelconque comprise entre 4 et 64 ko.
- Si la plupart des fichiers sont plusieurs fois plus grands que le paramètre **–blocksize**, donnez au paramètre **–fragsize** la même valeur que celle de **–blocksize**. Vous utiliserez sans doute plus d'espace disque, mais fluidifierez le traitement.
- Si l'agrégat est inférieur à 100 Mo, utilisez le paramètre **–logsize** pour vérifier que le journal est plus grand que la valeur par défaut (1 % de la taille de l'agrégat). En général, **logsize** ne doit jamais être inférieur à 1 000 blocs.

Chapitre 11. Analyse des performances avec l'utilitaire de suivi

L'utilitaire de suivi AIX est un puissant outil d'observation du système. Il capture un flux séquentiel d'événements horodatés, fournissant un niveau de détail extrêmement fin sur l'activité du système. Les événements sont restitués dans l'ordre chronologique et dans leur contexte. L'utilitaire est ainsi un outil privilégié d'observation du système et de l'exécution des applications. D'autres outils sont axés sur les statistiques (taux d'utilisation du CPU, délai d'attente des E/S, etc.), l'utilitaire de suivi est lui plus spécifiquement dédié à l'analyse du pourquoi et du comment.

Le système d'exploitation est conçu pour offrir une visibilité générale des exécutions sur le système. Les utilisateurs peuvent étendre la visibilité dans leurs applications en insérant des événements complémentaires et en fournissant des règles de formatage.

Lors de la conception et de l'implantation de cet outil, il a été pris grand soin de veiller à l'efficacité de la collecte des données, de façon à perturber au minimum l'activité système. De ce fait, l'utilitaire est aussi adapté à l'analyse des performances qu'à l'identification des incidents.

Plus en savoir plus, reportez-vous aux rubriques :

- "Présentation de l'utilitaire de suivi"
- "Exemple d'utilisation de l'utilitaire de suivi"
- "Lancement et contrôle du suivi depuis la ligne de commande"
- "Lancement et contrôle du suivi depuis un programme"
- "Ajout d'événements de suivi"
- "Syntaxe des strophes du fichier de format du suivi"

Présentation de l'utilitaire de suivi

L'utilitaire de suivi est plus souple que les services classiques de surveillance du système, qui présentent les statistiques tenues à jour par le système. Dans ces services en effet, la réduction des données (conversion d'événements système en statistiques) est largement associée à l'instrumentation du système. Ainsi, nombre de systèmes tiennent à jour le délai minimal, maximal et moyen, d'exécution de la tâche A et autorisent l'extraction de cette information.

L'utilitaire de suivi AIX n'associe pas fermement réduction des données et instrumentation, mais fournit un flot d'articles événements de suivi (appelés plus simplement *événements*). Il est inutile de décider à l'avance des statistiques requises. La réduction des données est dans une large mesure indépendante de l'instrumentation. L'utilisateur peut décider de déterminer les délais d'exécution minimal, maximal et moyen de la tâche A à partir du flot d'événements. Mais il est également possible d'extraire le délai moyen d'exécution de la tâche A lorsqu'elle est appelée par le processus B ; ou encore le délai d'exécution de la tâche A lorsque les conditions XYZ sont remplies ; ou de calculer la déviation standard à l'exécution de la tâche A ; ou même de décider qu'une autre tâche, reconnue par un flot d'événements, est plus significative. Cette souplesse est sans prix lorsqu'il s'agit de diagnostiquer des problèmes de performance ou de fonctionnement.

Outre les informations détaillées qu'il fournit sur l'activité du système, l'utilitaire de suivi permet d'instrumenter les programmes d'application et de collecter leurs événements de suivi à l'instar des événements du système. Le fichier de suivi contient un enregistrement complet des activités du système et des applications, horodatées et présentées chronologiquement.

Limitation de la quantité de données collectées

L'utilitaire de suivi génère d'importantes quantités de données. Il est donc impossible de les capturer sur de longues périodes, sous peine de submerger l'unité de stockage.
Pour utiliser efficacement l'utilitaire, vous avez le choix :

- L'utilitaire de suivi peut être activé/désactivé de plusieurs façons pour capturer des bribes de l'activité système. Il est pratique de capturer ainsi des secondes ou des minutes d'activité en vue d'un post-traitement. Une durée de cet ordre est suffisante pour caractériser les transactions principales au niveau des applications ou les passages intéressants d'une longue tâche.
- L'utilitaire de suivi peut être configuré pour acheminer le flot d'événements vers la sortie standard. Ce qui permet à un processus en temps réel de se connecter au flux d'événements et d'effectuer la réduction des données au fur et à mesure de la consignation des événements, créant ainsi une possibilité de surveillance à long terme. Une extension logique pour l'instrumentation spécialisée est de diriger le flot de données vers une unité auxiliaire susceptible soit de stocker d'importantes quantités de données, soit d'assurer la réduction dynamique des données. Cette technique est utilisée par les outils de performance **tprof**, **netpmon** et **filemon**.

Lancement et contrôle du suivi

Vous pouvez exploiter l'utilitaire de suivi selon trois modes :

- Mode sous-commande. Le suivi est lancé par une commande shell (**trace**) et prolongé par un dialogue avec l'utilisateur via des sous-commandes. Le travail objet du suivi doit être fourni par d'autres processus, car le processus shell d'origine est en cours d'utilisation.
- Mode commande. Le suivi est lancé par une commande shell (**trace –a**) qui comporte un indicateur spécifiant que l'utilitaire doit être exécuté en mode asynchrone. Le processus shell d'origine est disponible pour exécuter des commandes standard, entrecoupées de commandes de contrôle de suivi.

- Mode contrôlé par application. Le suivi est lancé (avec **trcstart()**) et contrôlé par des appels de sous-routines (telles que **trcon()**, **trcoff()**), à partir d'un programme d'application.

Formatage des données de suivi

Un utilitaire passe-partout de compte rendu de suivi est fourni par la commande **trcrpt**. Cet utilitaire offre peu de réduction de données, mais convertit le flot brut d'événements binaires en listing ASCII lisible. Les données peuvent être visuellement extraites par un lecteur, mais vous pouvez également développer des outils pour une meilleure réduction des données.

L'utilitaire affiche texte et données pour chaque événement en fonction des règles spécifiées dans le fichier de format du suivi. Le fichier de format du suivi par défaut est **/etc/trcfmt**. Ce fichier contient une strophe par ID événement, qui indique à l'utilitaire de compte rendu les règles de formatage pour cet événement. Cette technique permet aux utilisateurs d'ajouter leurs propres événements aux programmes et d'insérer les strophes correspondantes dans le fichier de format – pour préciser le format souhaité pour ces nouveaux événements.

Consultation des données de suivi

Lors du formatage des données de suivi, toutes les données relatives à un événement sont généralement placées sur une seule ligne. Des compléments d'information se trouvent éventuellement sur d'autres lignes. Selon les champs inclus, les lignes formatées peuvent facilement dépasser 80 caractères : mieux vaut prévoir une unité de sortie 132 colonnes.

Exemple d'utilisation de l'utilitaire de suivi

Obtention d'un fichier de suivi échantillon

Les données de suivi s'accumulent rapidement. Nous souhaitons restreindre la collecte aux données qui présentent un réel intérêt. Une technique possible consiste à émettre plusieurs commandes sur la même ligne de commande. Par exemple :

```
$ trace -a -k "20e,20f" -o ./trc_raw ; cp ../bin/track /tmp/junk ; trcstop
```

capture l'exécution de la commande **cp**. Nous avons utilisé deux caractéristiques de la commande de suivi. L'option `-k "20e,20f"` élimine la collecte des événements issus des fonctions **lockl** et **unlockl**. Ces appels sont nombreux et augmentent la taille du compte rendu sans apporter d'informations au niveau qui nous intéresse. L'option `-o ./trc_raw` entraîne l'écriture du fichier brut de sortie du suivi dans notre répertoire local.

> **Remarque :** Cet exemple est plus parlant si le fichier d'entrée ne se trouve pas déjà dans la mémoire cache du système. Choisissez comme fichier source un fichier d'environ 50 ko qui n'a fait l'objet d'aucun accès récent.

Formatage de l'échantillon de suivi

Pour notre compte rendu, nous avons utilisé la commande **trcrpt** sous la forme :

```
$ trcrpt -O "exec=on,pid=on" trc_raw > cp.rpt
```

Cette commande génère à la fois le nom complet qualifié du fichier exécuté (via **exec**) et l'ID processus qui lui est affecté.

Un coup d'œil sur le compte rendu indique la présence de nombreuses affectations de pages VMM et de suppressions d'événements dans le suivi, telle la séquence :

```
1B1 ksh     8525    0.003109888    0.162816    VMM page delete:      V.S=00
00.150E ppage=1F7F

                                                          delete_in_progress
process_private working_storage


1B0 ksh     8525    0.003141376    0.031488    VMM page assign:      V.S=00
00.2F33 ppage=1F7F

                                                          delete_in_progress
process_private working_storage
```

L'activité VMM ne nous intéressant pas à ce niveau, nous avons reformaté le suivi via :

```
$ trcrpt -k "1b0,1b1" -O "exec=on,pid=on" trc_raw > cp.rpt2
```

L'option `-k "1b0,1b1"` supprime les événements VMM superflus de la sortie formatée. Elle évite de relancer le suivi du travail pour éliminer les événements non souhaités. Nous aurions pu faire appel à la fonction **–k** de **trcrpt** au lieu de celle de la commande **trace** pour éliminer les événements **lockl** et **unlockl**, si nous avions eu besoin de consulter l'activité de verrouillage à un moment donné. Si notre intérêt ne portait que sur un petit nombre d'événements, nous aurions spécifié **–d "***id-ancre1,id-ancre2***"** pour générer un compte rendu ne portant que sur ces événements. L'ID ancre étant la première colonne du compte rendu, il est très rapide de compiler une liste d'ancres à inclure ou exclure.

Une liste complète des ID d'ancre de suivi se trouve dans **/usr/include/sys/trchkid.h.**

Interprétation d'un compte rendu de suivi

L'en-tête du compte rendu de suivi indique la date et le lieu où a été effectué le suivi, ainsi que la commande utilisée pour le générer :

```
Fri Nov 19 12:12:49 1993
System: AIX ptool Node: 3
Machine: 000168281000
Internet Address: 00000000 0.0.0.0
trace -ak 20e 20f -o -o ./trc_raw
```

Le corps du compte rendu, si affiché en caractères suffisamment petits, est semblable à :

```
ID PROCESS NAME  PID    ELAPSED_SEC   DELTA_MSEC  APPL SYSCALL KERNEL INTERRUPT
101 ksh         8525  0.005833472  0.107008         kfork
101 ksh         7214  0.012820224  0.031744         execve
134 cp          7214  0.014451456  0.030464         execcp../bin/trk/junk
```

Dans `cp.rpt`, vous pouvez observer :

- Les activités **fork**, **exec** et défaut de page du processus **cp.**
- L'ouverture du fichier d'entrée pour la lecture et la création du fichier `/tmp/junk`.
- Les appels système **read**/**write** successifs pour effectuer la copie.
- Le blocage du processus **cp** dans l'attente de la fin des E/S et de la distribution du processus **wait**.
- Les translations des requêtes de volume logique en requêtes de volume physique.
- Le mappage des fichiers (et non leur stockage dans les tampons classiques du noyau) et les accès en lecture provoquant des défauts de page qui doivent être résolus par VMM (Virtual Memory Manager).
- La détection des accès séquentiels par le gestionnaire de mémoire virtuelle (VMM), lequel déclenche un accès anticipé aux pages de fichier.
- La taille des données lues par anticipation – qui augmente avec la poursuite de l'accès séquentiel.
- Lorsque possible, la fusion par le pilote d'unité de disque des requêtes de fichier multiples en une seule requête d'E/S à destination du pilote.

La sortie du suivi semble au départ quelque peu surchargée. C'est un bon exercice d'apprentissage : si vous parvenez à discerner les activités décrites, vous êtes sur la bonne voie, et vous pourrez bientôt suffisamment maîtriser l'utilitaire de suivi et l'exploiter pour diagnostiquer les problèmes de performance du système.

Filtrage du compte rendu de suivi

L'intégralité des données de suivi n'est pas forcément utile. Vous pouvez décider des événements qu'il vous semble intéressant de suivre. Il est parfois utile, par exemple, de déterminer le nombre d'occurrences d'un événement donné. Pour répondre à la question "Combien de **open** se sont produits dans l'exemple de copie ?", recherchez d'abord l'ID événement de l'appel système **open**. Procédez comme suit :

```
$ trcrpt -j | grep -i open
```

Vous constatez que l'ID 15b correspond à l'événement **open**. Traitez maintenant les données de l'exemple :

```
$ trcrpt -d 15b -O "exec=on" trc_raw
```

Le compte rendu est inscrit sur la sortie standard et vous pouvez déterminer le nombre de sous-routines **open** exécutées. Si vous souhaitez vous limiter aux sous-routines **open** exécutées par le processus **cp**, relancez la commande comme suit :

```
$ trcrpt -d 15b -p cp -O "exec=on" trc_raw
```

Lancement et contrôle du suivi depuis la ligne de commande

L'utilitaire de suivi est configuré et la collecte des données est éventuellement lancée par la commande **trace**, dont la syntaxe est décrite dans le manuel *AIX Commands Reference*.

Une fois le suivi configuré par la commande **trace**, des contrôles permettent d'activer et de désactiver la collecte des données, et d'arrêter le suivi (l'arrêt déconfigure le suivi et libère les tampons). Ces contrôles sont appelés par : des sous-commandes, des commandes, des sous-routines ou des appels **ioctl**. La sous-routine et les interfaces **ioctl** sont décrites à "Lancement et contrôle du suivi depuis un programme", page 11-7.

Contrôle du suivi en mode sous-commande

Si la routine de suivi est configurée sans option **–a**, elle est exécutée en mode sous-commande. Une invite ">" remplace l'invite habituelle. Dans ce mode, sont reconnues les sous-commandes :

trcon	Lance ou relance la collecte des données d'événements.
trcoff	Suspend la collecte des données d'événements.
q ou quit	Arrête la collecte des données d'événements et met fin à la routine de suivi.
!*commande*	Exécute la *commande* shell spécifiée.

Contrôle du suivi par commandes

Si la routine de suivi est configurée pour une exécution asynchrone (**trace –a**), le suivi peut être contrôlé par les commandes :

trcon	Lance ou relance la collecte des données d'événements.
trcoff	Suspend la collecte des données d'événements.
trcstop	Arrête la collecte des données d'événements et met fin à la routine de suivi.

Lancement et contrôle du suivi depuis un programme

L'utilitaire de suivi AIX peut être lancé à partir d'un programme, via un appel de sous-routine. Cette sous-routine, **trcstart**, se trouve dans la bibliothèque **librts.a**. La syntaxe de la sous-routine **trcstart** est :

```
int trcstart(char *args)
```

args étant la liste des options que vous auriez associées à la commande **trace**. Par défaut, le suivi système (canal 0) est lancé. Pour lancer un suivi générique, spécifiez l'option **–g** dans la chaîne *args*. L'exécution terminée, la sous–routine**trcstart** renvoie l'ID canal. Pour les suivis génériques, cet ID canal permet d'enregistrer dans le canal générique privé.

Si vous compilez un programme via cette sous-routine, le lien à la bibliothèque **librts.a** doit être explicitement demandé (spécifiez **–l rts** comme option de compilation).

Contrôle du suivi via les sous-routines de suivi

Le contrôle de la routine de suivi s'effectue via des sous-routines de la bibliothèque **librts.a**. Ces sous-routines renvoient zéro lorsque la commande aboutit. Il s'agit des sous-routines :

int trcon()	Lance ou relance la collecte des données de suivi.
int trcoff()	Suspend la collecte des données de suivi.
int trcstop()	Arrête la collecte des données de suivi et met fin à la routine de suivi.

Contrôle du suivi via des appels ioctl

Chacune des sous-routines de contrôle de suivi ci-dessus :

- ouvre (**open**) l'unité de contrôle de suivi (/**dev**/**systrctl**),
- émet l'**ioctl** approprié,
- ferme (**close**) l'unité de contrôle,
- revient au programme appelant.

Pour activer/désactiver le suivi de portions individuelles de code, il peut être plus efficace pour le programme d'émettre directement les contrôles **ioctl**. Ce qui évite la succession d'ouvertures et de fermetures de l'unité de contrôle du suivi. Pour utiliser l'interface **ioctl** dans un programme, ajoutez **<sys/trcctl.h>** pour définir les commandes **ioctl**. La syntaxe de **ioctl** est la suivante :

```
ioctl (fd,CMD,Canal)
```

où :

fd	est le descripteur de fichier obtenu par l' ouverture de /**dev**/**systrctl**
CMD	est : TRCON, TRCOFF ou TRCSTOP
canal	est le canal de suivi (0 pour le suivi système).

L'exemple suivant illustre comment lancer le suivi à partir d'un programme, le suivi ne portant que sur une portion spécifiée du code :

```
#include <fcntl.h>
#include <sys/trcctl.h>
extern int trcstart(char *arg);
char *ctl_dev ="/dev/systrctl";
int ctl_fd;
main()
{
     printf("configuring trace collection \n");
     if (trcstart("-ad")){
         perror("trcstart");
         exit(1);
     }

     printf("opening the trace device  \n");
     if((ctl_fd =open (ctl_dev,O_RDWR))<0){
         perror("open ctl_dev");
         exit(1);
     }

     printf("turning data collection on \n");
     if(ioctl(ctl_fd,TRCON,0)){
         perror("TRCON");
         exit(1);
     }

     /* *** code here will be traced *** */
     printf("The code to print this line will be traced.");

     printf("turning data collection off\n");
     if (ioctl(ctl_fd,TRCOFF,0)){
         perror("TRCOFF");
         exit(1);
     }

     printf("stopping the trace daemon \n");
     if (trcstop(0)){
         perror("trcstop");
         exit(1);
     }

     exit(0);
}
```

Aucun fichier de sortie n'étant spécifié dans le paramètre pour la sous-routine **trcstart()**, la sortie du suivi se trouvera dans **/var/adm/ras/trcfile**, qui est également le fichier d'entrée par défaut de la commande **trcrpt**.

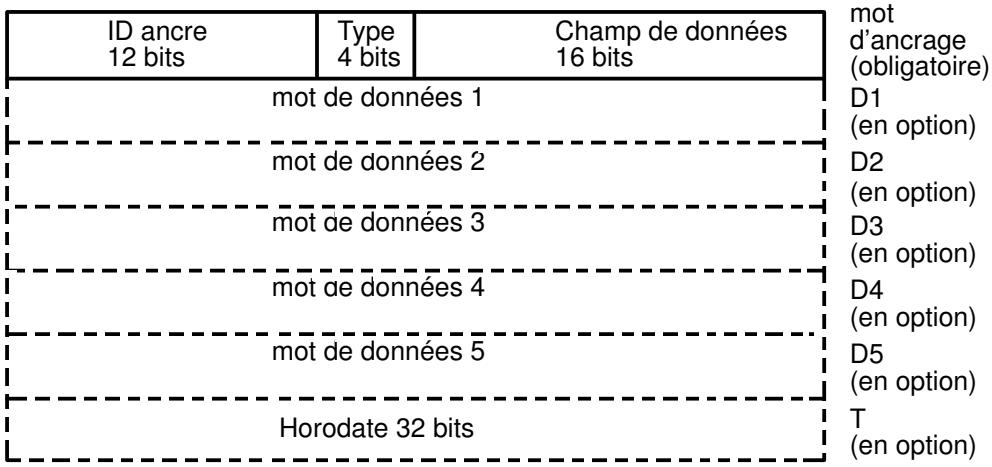
Ajout d'événements de suivi

Le système d'exploitation est livré avec des événements clés. Il suffit à l'utilisateur d'activer le suivi pour capturer le flux d'événements système. Les développeurs peuvent instrumenter leur code application au cours du développement, à des fins d'optimisation. Ils disposent ainsi d'une vision claire des interactions entre leurs applications et le système.

Pour ajouter un événement de suivi, il vous faut définir les articles de suivi générés par votre programme, en fonction des conventions de l'interface de suivi. Insérez ensuite des macros d'ancrage de suivi aux endroits souhaités dans votre programme. Le suivi peut être lancé par n'importe lequel des moyens d'appel et de contrôle standard (commandes, sous-commandes ou appels de sous-routines). Pour formater le suivi via le programme **trcrpt**, insérez, dans le fichier de format de suivi, des strophes pour décrire chaque nouvel article de suivi et préciser son formatage.

Formes d'un article événement de suivi

Un événement de suivi peut prendre diverses formes. Un événement est constitué d'un mot d'ancrage, de mots de données (facultatifs) et d'une horodate (facultatif), comme illustré à la figure "Format d'un article de suivi". Un type 4 bits est défini pour chaque forme que peut prendre un article. Le champ type est imposé par la routine d'enregistrement pour que l'utilitaire de compte rendu puisse toujours passer d'un événement à l'autre pendant le traitement des données, même si les règles de formatage du fichier de format de suivi sont incorrectes ou absentes pour cet événement.



Format d'un article de suivi

Un article de suivi doit être le plus court possible. Nombre d'événements système se contentent du mot d'ancrage et de l'horodate. Un autre événement cité doit être utilisé avec parcimonie car moins efficace et plus gênant : il s'agit d'un format long permettant d'enregistrer des données de longueur variable. Dans cette forme, le champ de données 16 bits du mot d'ancrage est converti en un champ *longueur* qui décrit la longueur de l'enregistrement de l'événement.

Canaux de suivi

L'utilitaire de suivi peut prendre en charge simultanément jusqu'à huit canaux d'activité d'ancrage de suivi, numérotés de 0 à 7. Le canal 0, toujours utilisé pour les événements système, ne leur est néanmoins pas réservé : des applications peuvent y faire appel. Les sept autres canaux, appelés *canaux génériques*, sont destinés au suivi des activités des applications.

Au lancement du suivi, le canal 0 est utilisé par défaut. Une commande **trace –***n* (*n* étant le numéro de canal) lance le suivi vers un canal générique. L'usage des canaux génériques souffre quelques restrictions :

- L'interface vers les canaux génériques consomme plus de temps CPU que l'interface vers le canal 0 : d'une part, il faut distinguer entre les canaux et, d'autre part, les canaux génériques enregistrent des articles de longueur variable.
- Les événements enregistrés sur le canal 0 et sur les canaux génériques ne peuvent être corrélés que par le biais de l'horodate, et non séquentiellement : il peut donc y avoir des cas où il est impossible de déterminer l'antériorité d'un événement.

Macros d'enregistrement des événements de suivi

Il existe une macro pour chaque type possible d'enregistrement d'événement. Les macros sont définies dans **/usr/include/sys/trcmacros.h** et les ID événements, dans **/usr/include/sys/trchkid.h**. Ces deux fichiers doivent être inclus dans tout programme enregistrant des événements de suivi.

Les macros enregistrant les événements sur le canal 0 avec horodate sont les suivantes :

TRCHKL0T(*hw***)**

TRCHKL1T(*hw,D1***)**

TRCHKL2T(*hw,D1,D2***)**

TRCHKL3T(*hw,D1,D2,D3***)**

TRCHKL4T(*hw,D1,D2,D3,D4***)**

TRCHKL5T(*hw,D1,D2,D3,D4,D5***)**

Pour enregistrer les événements sur le canal 0 sans horodate, vous disposez de :

TRCHKL0(*hw***)**

TRCHKL1(*hw,D1***)**

TRCHKL2(*hw,D1,D2***)**

TRCHKL3(*hw,D1,D2,D3***)**

TRCHKL4(*hw,D1,D2,D3,D4***)**

TRCHKL5(*hw,D1,D2,D3,D4,D5***)**

Le champ *type* de l'enregistrement de l'événement de suivi est défini à la valeur correspondant à la macro utilisée, quelle que soit la valeur de ces 4 bits dans le paramètre *hw.*

Il n'existe que deux macros pour enregistrer les événements sur un des canaux génériques (1 à 7). Il s'agit de :

TRCGEN(*ch,hw,D1,len,buf***)**

TRCGENT(*ch,hw,D1,len,buf***)**

Ces macros enregistrent, dans le flux d'événements spécifié par le paramètre canal (*ch*), un mot d'ancrage (*hw*), un mot de données (*D1*) et *len* octets à partir du segment de données utilisateur commençant à l'emplacement spécifié par *buf*.

ID d'événements

L'ID événement d'un article de suivi identifie cet article comme appartenant à une classe définie. L'ID événement est la base sur laquelle le mécanisme de suivi enregistre ou ignore les ancrages de suivi, et sur laquelle la commande **trcrpt** inclut ou exclut les articles de suivi dans le compte rendu formaté.

Les ID événement sont sur 12 bits (trois chiffres hexadécimaux). Il en existe 4096 possibles. Les ID qui sont maintenus en permanence et livrés avec le code sont affectés de façon

permanente par BULL, pour éviter toute duplication. Pour permettre aux utilisateurs de définir des événements dans leurs environnements ou au cours de développements, l'intervalle d'ID compris entre 010 à 0FF (hexa) leur a été réservé : les utilisateurs en disposent à leur guise *dans leur propre environnement* (c'est-à-dire, dans tout ensemble de systèmes dans lequel ils sont prêts à assurer qu'un même ID événement n'est pas utilisé de façon ambiguë).

Attention : Il est important que les utilisateurs qui se servent de cet intervalle d'événements maintiennent le code dans leur environnement. Si vous livrez du code, instrumenté avec des ID d'ancrage temporaires, dans un environnement dans lequel vous ne contrôlez pas l'utilisation des ID, vous risquez des collisions avec d'autres programmes qui utilisent les mêmes ID dans cet environnement.

Les ID événement doivent être conservés car peu nombreux, mais ils peuvent être étendus par le biais du champ de données 16 bits. Il existe donc 65536 événements distincts possibles pour chaque ID d'ancrage théorique. La seule raison d'avoir un ID unique est qu'un ID est le niveau auquel la collecte et le rapport de filtrage sont disponibles dans l'utilitaire de suivi.

Un événement ajouté par l'utilisateur peut être formaté via la commande **trcrpt** s'il existe une strophe pour l'événement dans le fichier de format de suivi spécifié. Ce fichier est un fichier ASCII modifiable : reportez-vous à "Syntaxe des strophes du fichier de format de suivi", page 11-13.

Exemples de codage et de formatage d'événements

L'exemple suivant illustre l'utilisation d'événements de suivi pour chronométrer la durée d'exécution d'une boucle :

```
#include <sys/trcctl.h>
#include <sys/trcmacros.h>
#include <sys/trchkid.h>
char *ctl_file = "/dev/systrctl";
int ctlfd;
int i;
main()
{
  printf("configuring trace collection \n");
  if (trcstart("-ad")){
    perror("trcstart");
    exit(1);
  }

  printf("opening the trace device  \n");
  if((ctlfd = open(ctl_file,0))<0){
    perror(ctl_file);
    exit(1);
  }

  printf("turning  trace on \n");
  if(ioctl(ctlfd,TRCON,0)){
    perror("TRCON");
    exit(1);
  }
```

```
  for(i=1;i<11;i++){
    TRCHKL1T(HKWD_USER1,i);

    /* The code being measured goes here. The interval */
    /* between occurrences of HKWD_USER1 in the trace  */
    /* file is the total time for one iteration.       */
  }


  printf("turning trace off\n");
  if(ioctl(ctlfd,TRCSTOP,0)){
    perror("TRCOFF");
    exit(1);
  }

  printf("stopping the trace daemon \n");
  if (trcstop(0)){
    perror("trcstop");
    exit(1);
  }

  exit(0);
}
```

Lorsque vous compilez ce programme, vous devez établir un lien à la bibliothèque **librts.a** comme suit :

```
$ xlc -O3 sample.c -o sample -l rts
```

HKWD_USER1 a pour ID 010 (hexadécimal). Vous pouvez le vérifier en consultant **/usr/include/sys/trchkid.h**.). L'utilitaire de compte rendu ignore comment formater l'événement HKWD_USER1, sauf si des règles sont indiquées dans le fichier de format de suivi. L'exemple de strophe suivante peut être utilisée :

```
# User event HKWD_USER1 Formatting Rules Stanza
# An example that will format the event usage of the sample program
010 1.0 L=APPL "USER EVENT - HKWD_USER1" O2.0      \n \
               "The # of loop iterations =" U4     \n \
               "The elapsed time of the last loop = " \
                endtimer(0x010,0x010) starttimer(0x010,0x010)
```

N'insérez pas cette strophe dans le fichier **/etc/trcfmt** original, mais faites une copie de celui-ci dans votre répertoire (`mytrcfmt`, par exemple). Lorsque vous exécutez le programme exemple, les données d'événement brutes sont capturées dans le fichier journal par défaut, puisqu'aucun autre fichier n'a été spécifié à la sous-routine **trcstart**. Vous souhaitez probablement filtrer la sortie pour ne conserver que vos événements. Pour ce faire, lancez la commande **trcrpt**, comme suit :

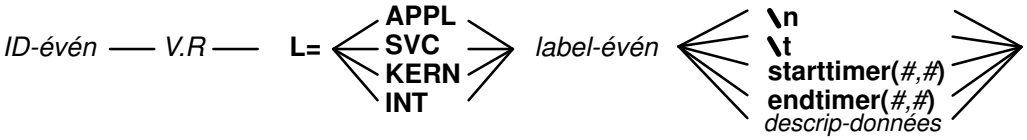
```
trcrpt -d 010 -t mytrcfmt -O "exec=on" > sample.rpt
```

Vous pouvez parcourir `sample.rpt` pour voir le résultat.

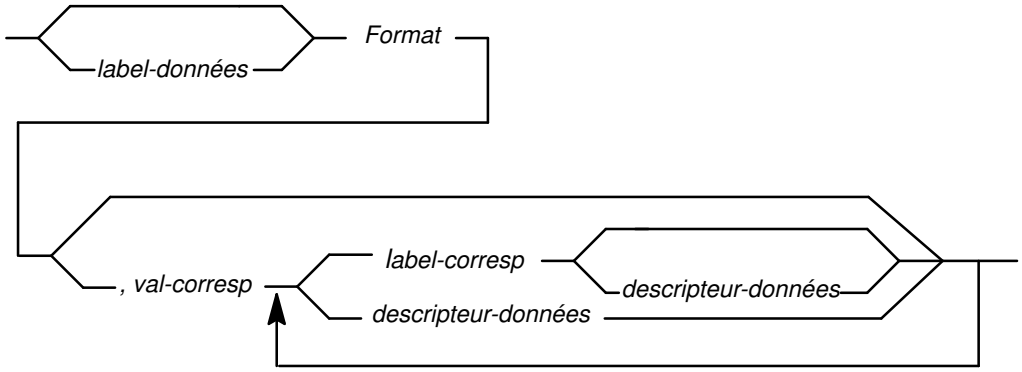
Syntaxe des strophes du fichier de format de suivi

L'objet du fichier de format de suivi est de fournir des règles de présentation et d'affichage des données attendues pour chaque ID d'événement. Ce qui permet de formater les nouveaux événements sans modifier l'utilitaire de compte rendu. Des règles sont simplement ajoutées au fichier de format pour ces événements. La syntaxe des règles offre une grande souplesse de présentation des données.

La figure "Syntaxe d'une strophe d'un fichier de format de suivi", page 11-13, illustre la syntaxe pour un événement donné. Une strophe de format peut être aussi longue que nécessaire pour décrire les règles applicables à un événement donné. Si elle se prolonge sur plusieurs lignes, terminez les lignes intermédiaires par le caractère "\". Les champs sont les suivants :



où *descripteur-données* a la syntaxe :



Syntaxe d'une strophe d'un fichier de format de suivi

ID-évén	Chaque strophe commence par l'ID (3 chiffres hexadécimaux) de l'événement qu'elle décrit, suivi d'un espace.
V.R	Décrit la version (*V*) et la release (*R*) dans lesquels l'événement a été affecté en premier. Vous pouvez indiquer n'importe quels entiers pour *V* et *R* (un utilisateur peut vouloir conserver son propre mécanisme de suivi).
L=	Spécifie le niveau d'indentation du texte. La description textuelle d'un événement peut commencer à divers niveaux d'indentation. Ceci pour améliorer la lisibilité du compte rendu. Ces niveaux correspondent au niveau d'exécution du système. Les niveaux reconnus sont : niveau application (**APPL**), appel système de transition (**SVC**), niveau noyau (**KERN**) et interruption (**INT**).
label-évén	Chaîne de texte ASCII décrivant l'utilisation globale de l'ID événement – utilisée par l'option **–j** de la commande **trcrpt** pour fournir une liste d'événements avec leur premier niveau de description. Le *label-évén* apparaît également dans la sortie formatée de l'événement, sauf si le *label-évén* commence par le caractère @.

\n La strophe événement décrit comment analyser, libeller et présenter les données d'un enregistrement d'événement. La fonction **\n** (nouvelle ligne) peut être imbriquée dans la strophe événement pour forcer la présentation des données sur une nouvelle ligne. Les données peuvent ainsi être présentées sur plusieurs lignes.

\t Insère une tabulation à l'endroit où il se trouve dans l'analyse de la description. Cette fonction est semblable à la fonction **\n**. Des espaces peuvent également être insérés via les champs*label_données* ou *label_corresp*.

starttimer(*timerID***), endtimer(***timerID***)**

timerID est un identificateur unique qui associe un **starttimer** particulier avec un **endtimer** ultérieur ayant le même identificateur. Par convention (non obligatoire), *timerID* est de la forme :

ID de l'événement de départ, ID de l'événement final

Lorsque l'utilitaire de compte rendu rencontre une directive **starttimer** au cours de l'analyse d'un événement, il associe l'heure de l'événement de départ au *timerID* spécifié. Lorsqu'il rencontre un **endtimer** avec le même *timerID*, il affiche l'écart de temps (entre parenthèses) entre les deux événements. Les événements début et fin d'appel système font appel à cette fonction : au retour d'un événement appel système, la durée de cet appel est indiquée.

descrip-données

Décrit comment l'utilitaire de compte rendu doit consommer, libeller et présenter les données. La syntaxe du champ *descripteur-données* est développée dans la seconde partie de la figure "Syntaxe d'une strophe d'un fichier de format de suivi". Les différents champs du *descripteur-données* sont décrits ci-après :

format

L'utilisateur peut se représenter l'utilitaire de compte rendu comme doté d'un pointeur sur la partie données d'un événement. Ce pointeur est initialisé pour pointer sur le début des données d'événement (le champ de données 16 bits dans le mot d'ancrage). Le champ *format* décrit la quantité de données consommées par l'utilitaire à partir de ce point et comment considérer ces données. Par exemple, un champ *format* de **B***m.n* indique à l'utilitaire de consommer *m* octets et *n* bits de données et de les considérer comme binaires. (Les champs de format possibles sont décrits plus avant.) Si le champ de format n'est pas suivi d'une virgule, l'utilitaire génère les données consommées dans le format spécifié. S'il l'est, cela signifie que les données ne doivent pas être affichées, mais comparées aux *valeurs-corresp* qui suivent. Le descripteur de données associé avec la *valeur-corresp* correspondante étant ensuite appliqué au reste des données.

label-données

Le *label-données* est une chaîne ASCII précédant éventuellement la sortie des données consommées par le champ de format.

valeur-corresp

La *valeur-corresp* correspond à des données dont le format est identique à celui décrit par les champs de format précédents. Généralement, plusieurs *valeurs-corresp* suivent un champ de format en cours de recherche de correspondance. Les champs successifs de correspondance sont séparés par des virgules. La dernière valeur n'est pas suivie d'une virgule. Un * remplace n'importe quelle chaîne de caractères. Un caractère générique est souvent utilisé comme dernier champ *valeur-corresp* pour spécifier les règles par défaut, si aucune correspondance n'a été trouvée avec les champs *valeur-corresp* précédents.

label-corresp

Le *label-corresp* est une chaîne ASCII, générée pour la correspondance concernée.

Tous les champs de format possibles sont décrits dans les commentaires du fichier /**etc**/**trcfmt**. En voici une brève présentation.

Descriptions de champs de format

A*m.n*	Spécifie que *m* octets de données doivent être consommées comme texte ASCII et que le texte doit être affiché dans un champ de sortie de *n* caractères. Le pointeur de données est déplacé de *m* octets.
S1, S2, S4	Spécifie une chaîne alignée à gauche. La longueur du champ est définie comme égale à 1 octet (**S1**), 2 octets (**S2**) ou 4 octets (**S4**). Le pointeur de données est déplacé en conséquence.
B*m.n*	Spécifie des données binaire de *m* octets et *n* bits. Le pointeur de données est déplacé en conséquence.
X*m*	Spécifie des données hexadécimales de *m* octets. Le pointeur de données est déplacé en conséquence.
D2, D4	Spécifie un format décimal signé. Des données de longueur 2 octets (**D2**) ou 4 octets (**D4**) sont consommées.
U2, U4	Spécifie un format décimal non signé. Des données de longueur 2 ou 4 octets sont consommées.
F4, F8	Virgule flottante de 4 ou 8 octets.
G*m.n*	Positionne le pointeur de données *m* octets et *n* bits à l'intérieur des données.
O*m.n*	Omet, à partir de la position courante du pointeur de données, les *m* octets et les *n* bits suivants.
R*m*	Recule le pointeur de données de *m* octets.

Quelques macros peuvent servir de champs de format, pour un accès rapide aux données. Par exemple :

$HD, $D1, $D2, $D3, $D4, $D5

Accède au champ de données 16 bits du mot d'ancrage et des mots de données 1 à 5 de l'enregistrement de l'événement, respectivement. Le pointeur de données n'est pas déplacé. Les données auxquelles accède une macro sont hexadécimales par défaut. Une macro peut être associée à un autre type de données (X, D, U, B) par le biais du caractère "%" suivi du nouveau code de format. Par exemple :

```
$D1%B2.3
```

Cette macro entraîne l'accès au mot de données 1, mais en le considérant comme 2 octets et 3 bits de données binaires.

Les commentaires intégrés au fichier /**etc**/**trcfmt** décrivent les autres possibilités de macros et de formats, et expliquent comment créer des macros personnalisées.

Chapitre 12. Outil de diagnostic PDT

PDT évalue l'état d'un système et surveille les modifications intervenant sur la charge de travail et les performances. Il tente d'identifier les problèmes à leur naissance, proposant des solutions avant qu'ils ne deviennent critiques. PDT est disponible à partir d'AIX version 4.1.

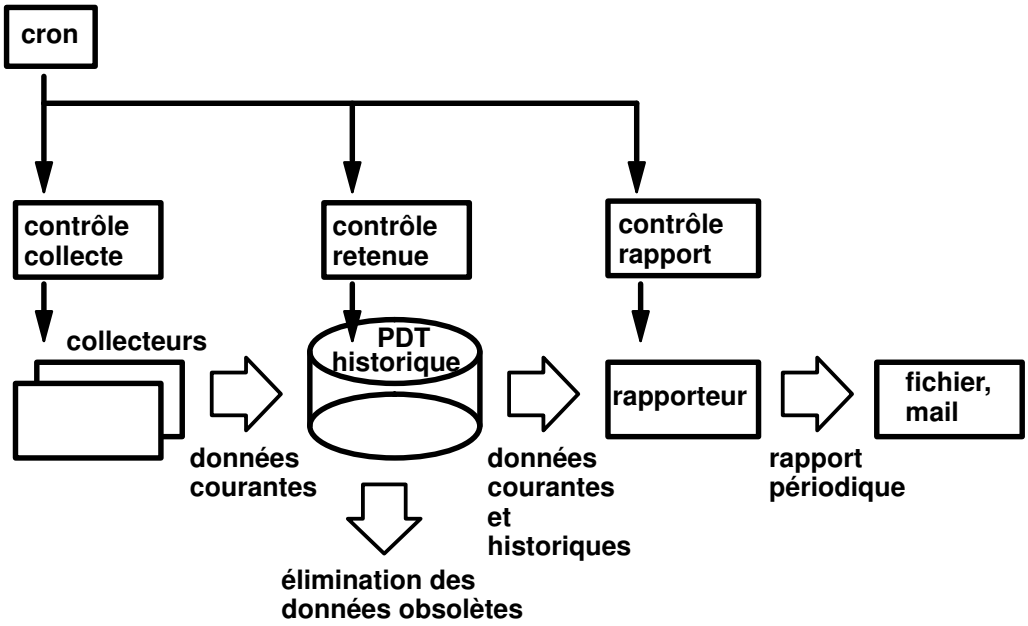
Pour l'essentiel, PDT ne requiert aucune entrée utilisateur. La collecte et le report des données PDT sont facilement activés et aucune intervention ultérieure de l'administrateur n'est requise. Les données sont régulièrement collectées et enregistrées en vue d'une analyse historique, et un rapport est généré et adressé à l'ID utilisateur `adm`. Normalement, seuls les problèmes apparents les plus significatifs sont consignés sur le rapport. L'absence de problème significatif est également signalée, le cas échéant. Vous pouvez personnaliser PDT pour que les rapports soient acheminés vers un autre utilisateur ou que les problèmes de gravité moindre soient également consignés.

Cette rubrique traite des points suivants :

- "Structure de PDT"
- "Portée de l'analyse PDT"
- "Exemple de compte rendu PDT"
- "Installation et activation de PDT"
- "Personnalisation de PDT"
- "Réponse aux messages PDT"

Structure de PDT

Comme illustré à la figure "Structure des composants de PDT", l'application PDT est constituée de trois composants :



Structure des composants de PDT

- L'élément de collecte, composé d'un ensemble de programmes qui collectent et enregistrent régulièrement les données.
- L'élément de validation qui passe régulièrement en revue les données collectées et élimine les données obsolètes.
- L'élément de consignation qui produit régulièrement un diagnostic à partir de l'ensemble courant des données historiques.

PDT prend en compte différents aspects de la configuration d'un système, sa disponibilté et ses performances, lors de de son estimation. Les zones de déséquilibre potentiel sont notamment mises en évidence (configuration des E/S, de la pagination, etc.), ainsi que d'autres problèmes de configuration (disques non affectés à des groupes de volume, par exemple). Il est procédé à une large évaluation, portant aussi bien sur la taille des fichiers et systèmes de fichiers, que sur l'utilisation de la zone de pagination, les délais imposés par le réseau ou ceux relatifs à la charge de travail, etc.

Portée de l'analyse PDT

PDT collecte quotidiennement les données relatives à la configuration, la disponibilité, la charge de travail et les performances. Ces données sont maintenues dans un enregistrement historique. Environ un mois de données sont ainsi conservées. PDT génère également quotidiennement un rapport de diagnostic. Le rapport est envoyé à l'utilisateur `adm`.

Une copie de ce rapport est en outre stockée dans **/var/perf/tmp/PDT_REPORT.** Au préalable, l'ancien rapport est renommé **/var/perf/tmp/PDT_REPORT.last.**

Bien que nombre de problèmes de performances soient de nature spécifique (mémoire insuffisante sur un système, par exemple), PDT tente d'appliquer quelques principes généraux dans sa recherche. En voici quelques-uns.

Exploitation équilibrée des ressources

D'une facon générale, s'il existe plusieurs ressources de même type, équilibrer leur exploitation génère une amélioration des performances.

- Nombre comparable de volumes physiques (disques) sur chaque carte de disque
- Espace de pagination réparti sur plusieurs volumes physiques
- Charge grosso modo équivalente sur les différents volumes physiques.

Circonscription des opérations

L'exploitation des ressources a ses limites. Toute tendance à les outrepasser doit être détectée et signalée.

- Une unité de disque ne peut être utilisée plus de 100 % du temps.
- Fichiers et systèmes de fichiers ne peuvent excéder l'espace qui leur est affecté.

Identification de la tendance de la charge de travail

Les tendances peuvent indiquer un changement de la nature de la charge de travail ou une augmentation de ressources utilisées :

- Nombre d'utilisateurs connectés.
- Nombre total de processus.
- Pourcentage de CPU disponible.

Opérations sans erreur

Les erreurs (matérielles et logicielles) ont souvent une incidence sur les performances :

- Consultez les journaux d'erreurs matérielles et logicielles.
- Signalez les pages VMM incorrectes.

Examen des modifications

De nouvelles charges de travail ou de nouveaux processus commençant à consommer des ressources sont parfois des signes avant-coureurs de problèmes.

- Apparence des processus consommant d'importantes ressources CPU ou mémoire.

Adéquation des paramètres système

Les paramètres système sont nombreux. Sont-ils tous définis de façon appropriée ?

- **maxuproc** a-t-il une valeur suffisante ?
- Qu'en est-il des paramètres de contrôle de la charge mémoire ?

Le rapport PDT est constitué de plusieurs sections (voir ci-après). Après l'en-tête, la section Alerts indique les violations identifiées des principes notés ci-dessus. En l'absence d'alertes, cette section ne figure pas dans le rapport. Les deux sections suivantes concernent les tendances en amont et en aval : elles sont axées sur l'anticipation des problèmes et non sur l'identification de problèmes existants. Les mêmes principes sont généralement appliqués – mais dans l'optique de prévoir les violations futures. Si aucune tendance de ce type (en amont ou en aval) n'est détectée, ces sections ne figurent pas dans le rapport.

Exemple de compte rendu PDT

```
 Performance Diagnostic Facility 1.0

 Report printed: Tue Aug 3 10:00:01 1993
 Host name: test.austin.ibm.com
 Range of analysis is from:  Hour 16 on Monday, July 5th, 1993
 to: Hour 9 on Tuesday, August 3rd, 1993.

[To disable/modify/enable collection or reporting, execute the
pdt_config script]

--------------------- Alerts ---------------------
  I/O BALANCE
   -  Phys. vol. hdisk0 is significantly busier than others
      volume cd0, mean util. = 0.00
      volume hdisk0, mean util. = 11.75
      volume hdisk1, mean util. = 0.00
  PAGE SPACE AND MEMORY
   -  Mean page space used = 46.85 MB
      System has 32MB memory; may be inadequate.
      Consider further investigations to determine if memory is a
bottleneck

------------------- Upward Trends ----------------
  FILE SYSTEMS
   -  File system hd2 (/usr) PERCENTAGE FULL
      now, 45.00 % full, and growing an avg. of 2.0 %/day
      At this rate, hd2 will be full in about 15 days
  PAGE SPACE
   -  Page space hd6 USE
      now, 44.80 MB and growing an avg. of 1.81 MB/day
      At this rate, hd6 will be full in about 30 days
  WORKLOAD TRACKING
   - Workload nusers indicator is increasing;
     now 23, and growing an avg. of 1.2 per day

----------------------- System Health ---------------
  SYSTEM HEALTH
   -  Current process state breakdown:
      2.00 [ 3.0 %] : waiting for the cpu
      64.00 [ 97.0 %] : sleeping
      66.00 = TOTAL
      [based on 1 measurement consisting of 10 2-second samples]

------------------ Summary -----------------------
  This is a severity level 2 report
  Further details are available at severity levels > 2
```

Dans cet exemple, la section d'en-tête indique le numéro de version de PDT, la date du rapport, l'hôte à partir duquel ont été collectées les données et les dates de début et de fin des données analysées.

La section suivante, Alerts, indique les conditions de configuration et de charge suspectes. Dans l'exemple, il apparaît qu'un des trois disques système récupère l'essentiel des activités d'E/S. Il est clair que la charge des E/S n'est pas répartie de façon à tirer le meilleur parti des ressources disponibles. Le message suivant,
PAGE SPACE AND MEMORY, suggère que le système est peut-être sous-configuré au niveau mémoire.

La section Upward Trends identifie deux tendances possibles. La première est que le système de fichiers sur le volume logique **hd2** (système de fichiers /**usr**) croît au rythme moyen de 2 % par jour. La date de saturation probable est indiquée, calculée sur la base d'une croissance linéaire continue.

La seconde tendance est l'apparente croissance systématique du niveau d'utilisation de l'une des zones de pagination. Des informations sur son taux de croissance et sur la date présumée de saturation sont données. Savoir quels sont les systèmes de fichiers et les espaces de pagination qui frôlent la saturation est de la plus haute importance (surtout si le taux de croissance est élevé ou que la date de saturation prévue est imminente), car leur saturation peut provoquer un blocage du système ou de l'application.

La troisième tendance est une modification d'un des indicateurs de charge de travail. Les indicateurs surveillés par PDT sont les suivants :

Mot-clé	Indicateur
nusers	Nombre total d'utilisateurs connectés.
loadavg	Moyenne de charge sur 15 minutes.
nprocesses	Nombre total de processus.
STAT_A	Nombre de processus actifs.
STAT_W	Nombre de processus permutés.
STAT_Z	Nombre de processus zombis.
STAT_I	Nombre de processus libres.
STAT_T	Nombre de processus arrêtés après réception d'un signal.
STAT_*x*	Nombre de processus signalés à l'état *x* par la commande **ps**, *x* étant un état non répertorié ci-dessus.
cp	Durée requise pour copier un fichier de 40 ko.
idle_pct_cpu0	Pourcentage de CPU disponible.
idle_pct_avg	Pourcentage de CPU disponible.

La section suivante, System Health, se sert d'un certain nombre d'indicateurs de charge pour déterminer l'usage que font les processus de leur temps.

La dernière section du compte rendu (Summary) indique le niveau de gravité sélectionné, et si changer de niveau donne accès à des informations supplémentaires. (Le niveau de gravité le plus élevé est 1, niveau par défaut du compte rendu. Le niveau le plus bas est 3.)

Tout message (à l'exclusion des informations de l'en-tête et du récapitulatif) apparaissant dans le compte rendu PDT doit être examiné. Vous devez remédier au problème signalé ou du moins trouver une explication à l'erreur. Les réponses possibles aux messages spécifiques sont indiquées à "Réponse aux messages PDT", page 12-10.

Installation et activation de PDT

PDT est installé via **installp** comme option **bos.perf.diag_tool** du BOS sous licence d'AIX version 4.1.

Pour démarrer la collecte et la consignation des données, PDT doit être activé : exécutez à cet effet le script **/usr/sbin/perf/diag_tool/pdt_config.** Seul l'utilisateur `root` est habilité à exécuter ce script. L'exécution terminée, le message suivant s'affiche :

```
# /usr/sbin/perf/diag_tool/pdt_config
 _________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number:
```

Si vous répondez 4, la collecte et la consignation par défaut sont activées. L'entrée crontab de l'utilisateur `adm` est mise à jour pour intégrer les entrées PDT. La collecte est effective lorsque les travaux cron sont exécutés par **cron**. Sélectionnez 7 pour mettre fin au programme **pdt_config**.

Pour désactiver la collecte, sélectionnez l'option 5.

Personnalisation de PDT

Certains aspects de PDT peuvent être personnalisés. N'importe quel utilisateur peut, par exemple, être désigné comme destinataire régulier des rapports PDT, et la durée de conservation des données dans l'enregistrement historique PDT peut être modifiée. Toute personnalisation suppose de modifier un des fichiers PDT dans **/var/perf/cfg/diag_tool/** ou d'exécuter le script **/usr/sbin/perf/diag_tool/pdt_config**.

Nous vous conseillons de ne rien modifier avant d'avoir obtenu et étudié quelques rapports PDT et d'avoir acquis une certaine aisance dans la manipulation de cet outil.

Désignation d'un autre destinataire du rapport et modification du niveau de gravité

Par défaut, les rapports sont générés avec un niveau de gravité de 1, ce qui signifie que seuls les problèmes les plus sérieux sont identifiés. Il existe deux autres niveaux (2, 3) offrant généralement plus de détails. Par ailleurs, tout rapport PDT est expédié vers l'ID utilisateur `adm` ; peut-être souhaitez-vous l'adresser à un autre utilisateur ou ne pas l'envoyer du tout.

Ces deux paramètres sont contrôlés par le script **pdt_config**. Le dialogue suivant désigne un nouvel utilisateur destinataire et modifie le niveau de gravité :

```
# /usr/sbin/perf/diag_tool/pdt_config

_________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 1

current PDT report recipient and severity level
adm 1

_________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 2
```

```
enter id@host for recipient of report : rsmith
enter severity level for report (1-3): 2

report recipient and severity level
rsmith 2

_________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 1

current PDT report recipient and severity level
rsmith 2

_________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 7
#
```

Dans cet exemple, l'utilisateur `rsmith` devient le destinataire et le niveau de gravité passe à 2. Ce qui signifie que l'utilisateur `rsmith` recevra le rapport PDT et que les messages de gravité 1 et 2 y seront inclus. Notez l'utilisation de l'option 1 pour déterminer le destinataire actuel du rapport PDT ainsi que le niveau de gravité consigné.

Pour mettre fin à la consignation (sans interrompre la collecte), l'option 3 est sélectionnée, par exemple :

```
#/usr/sbin/perf/diag_tool

_________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 3

disable PDT reporting done

_________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 1

reporting has been disabled (file .reporting.list not found).
```

```
_____________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 7
#
```

Niveaux de gravité PDT

Les listes suivantes indiquent les problèmes susceptibles de se poser à chaque niveau de gravité. N'oubliez pas que sélectionner une gravité *n* entraîne la consignation de tous les problèmes de gravité inférieure ou égale à *n.*

Problèmes de gravité 1

- Système de fichiers JFS indisponible
- Système de fichiers JFS quasi-saturé
- Volume physique non affecté à un groupe de volumes
- Espaces de pagination tous définis sur un seul volume physique
- Mémoire système apparemment insuffisante pour la charge de travail actuelle
- Espace de pagination quasi-saturé
- Problème possible au niveau des paramètres de contrôle de charge
- Trames de mémoire défectueuses détectées par VMM
- Hôte de **.nodes** inaccessible

Problèmes de gravité 2

- Déséquilibre dans la configuration des E/S (disques par carte, par exemple)
- Déséquilibre dans l'affectation de l'espace de pagination sur les volumes physiques concernés
- Fragmentation d'un espace de pagination dans un groupe de volumes
- Déséquilibre significatif de la charge des E/S sur les volumes physiques
- Nouveau processus identifié comme gros consommateur de mémoire ou de CPU
- Fichiers de **.files** présentant une croissance (ou un déclin) de taille systématique
- Système de fichiers ou espace de pagination présentant une croissance (ou un déclin) systématique au niveau de l'utilisation de l'espace
- Dégradation du temps de réponse **ping** ou du pourcentage de paquets perdus sur un hôte de **.nodes**
- Processus getty consommant trop de temps CPU
- Processus gros consommateur de CPU ou de mémoire présentant une croissance (ou un déclin) systématique au niveau de l'utilisation des ressources

Messages de gravité 3

- Les messages de gravité 3 fournissent des compléments aux messages de gravité 1 et 2, sur les problèmes identifiés aux niveaux 1 et 2 – caractéristiques de la collecte des données, tel le nombre d'échantillons, par exemple.

Obtention d'un rapport PDT à la demande

Outre le rapport périodique, un utilisateur quelconque peut demander un rapport à partir des données existantes via la commande **/usr/sbin/perf/diag_tool/pdt_report** [*niveau-gravité*]. Le rapport est produit avec le niveau de gravité spécifié (1 par défaut) et écrit dans **stdout**. Générer un rapport de cette manière n'a aucune incidence sur les fichiers **/var/perf/tmp/PDT_REPORT** ou **/var/perf/tmp/PDT_REPORT.last**.

Consignation d'erreurs PDT

Des erreurs peuvent advenir dans n'importe lequel des composants PDT. En général, une erreur ne met pas fin à PDT. Mais un message est envoyé vers le fichier d'erreur standard de PDT : **/var/perf/tmp/.stderr**, et cette phase de traitement se termine.

Les utilisateurs devant faire face à des situations inattendues, telle que l'absence de rapport, trouveront de précieuses indications dans le fichier **/var/perf/tmp/.stderr**.

Désinstallation de PDT

Il est impossible de désinstaller PDT directement via **pdt_config**, mais si l'option 6 est requise, un message explicite les étapes nécessaires au retrait de PDT du système :

```
# /usr/sbin/perf/diag_tool/pdt_config

_________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 6

  PDT is installed as package bos.perf.diag_tool in the bos lpp.
  Use the installp facility to remove the package

_________________PDT customization menu_____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 7
#
```

Modification de la liste des fichiers contrôlés par PDT

PDT analyse fichiers et répertoires à la recherche d'accroissement systématique de taille. Il n'examine que les fichiers et répertoires figurant dans le fichier **/var/perf/cfg/diag_tool/.files**. Le fichier **.files** comporte un nom de fichier/répertoire par ligne. Son contenu par défaut est :

```
/usr/adm/wtmp
/var/spool/qdaemon/
/var/adm/ras/
/tmp/
```

Vous pouvez modifier ce fichier via un éditeur pour surveiller des fichiers et répertoires importants pour votre système.

Modification de la liste des hôtes contrôlés par PDT

PDT surveille le délai **ping** moyen des hôtes listés dans **/var/perf/cfg/diag_tool/.nodes**. Ce fichier n'est pas livré avec PDT (ce qui signifie qu'aucune analyse d'hôte n'est effectuée

par défaut), mais peut être créé par l'administrateur. Le fichier **.nodes** comporte un nom d'hôte par ligne.

Modification de la durée de conservation de l'enregistrement historique

Régulièrement, un script shell de validation est exécuté, qui élimine de l'enregistrement historique PDT les données périmées. Toutes les données sont soumises à la même politique de conservation. Cette politique est décrite dans le fichier **/var/perf/cfg/diag_tool/.retention.list**. Le contenu par défaut de **.retention.list** est :

```
* * * 35
```

qui fixe à 35 jours le délai de conservation de toutes les données. Vous pouvez remplacer 35 par n'importe quel entier non signé.

PDT se sert de l'enregistrement historique pour évaluer les tendances et identifier les modifications du système. Augmenter le délai de conservation étend la portée de l'analyse, mais en augmente le coût en termes d'espace disque et de temps de traitement PDT.

L'enregistrement historique PDT est conservé dans **/var/perf/tmp/.SM**. Le script de validation crée une copie de ce fichier dans **/var/perf/tmp/.SM.last** avant d'effectuer l'opération de validation. Les données historiques éliminées sont par ailleurs ajoutées à **/var/perf/tmp/.SM.discards**.

L'existence de **/var/perf/tmp/.SM.last** fournit une sauvegarde limitée, mais l'administrateur doit vérifier que le fichier **/var/perf/tmp/.SM** est régulièrement sauvegardé. Si le fichier est perdu, PDT continue à fonctionner, mais sans informations d'historique. Au fil du temps, l'historique se reconstruit avec les nouvelles données collectées.

Modification de l'heure des collectes, des validations et de la génération des comptes rendus

La collecte, la consignation et la validation sont pilotées par trois entrées de la table **cron** de l'utilisateur `adm`. La collecte a lieu à 9 h tous les jours ouvrés, la consignation à 10 h ces mêmes jours, tandis que la validation est effectuée une fois par semaine, le samedi à 21 h. Les entrées cron (créées par exécution du script **/usr/sbin/perf/diag_tool/pdt_config** et sélection de l'option 2) sont indiquées ci-après :

```
0  9 * * 1-5   /usr/sbin/perf/diag_tool/Driver_ daily
0 10 * * 1-5   /usr/sbin/perf/diag_tool/Driver_ daily2
0 21 * * 6     /usr/sbin/perf/diag_tool/Driver_ offweekly
```

Il est possible de modifier ces heures en éditant la table **cron** de l'utilisateur `adm`, mais nous vous le déconseilllons fortement.

Réponse aux messages PDT

PDT identifie plusieurs types de problèmes. Les réponses à y apporter dépendent des ressources disponibles et des priorités de chaque structure. En voici quelques exemples.

Problème :	**Système de fichiers JFS indisponible**
Réponse :	Recherchez la cause de cette indisponibilité.
Comm. utiles :	**lsfs** (pour déterminer l'état du système de fichiers).

Problème :	**Système de fichiers JFS quasi-saturé**
Réponse :	Recherchez des fichiers volumineux dans le système de fichiers, générés peut-être par l'emballement d'un processus. Ce système de fichiers a-t-il présenté une tendance à croître sur une longue durée (vérifiez ce point dans le reste du rapport PDT – ou dans les anciens rapports) ?
Comm. utiles :	**du**, **ls**

Problème : **Volume physique non affecté à un groupe de volumes**

Réponse : Un volume doit être défini dans un groupe de volumes, faute de quoi, il est inaccessible à AIX et, de ce fait, perdu.

Comm. utiles : **lspv** (pour confirmer que le volume n'est pas affecté).

smit (pour manipuler les groupes de volume).

Problème : **Espaces de pagination tous définis sur un seul volume physique**

Réponse : Le système dispose de plusieurs volumes physiques, mais tous les espaces de pagination sont définis sur un seul volume. Si le système doit faire appel à la pagination, cette configuration entraîne une diminution des performances.

Comm. utiles : **smit** (pour modifier les espaces de pagination).

Problème : **Mémoire apparemment insuffisante pour la charge de travail actuelle**

Réponse : Si le système fait souvent appel à la pagination, il convient sans doute d'augmenter la mémoire pour améliorer les performances.

Comm. utiles : **lsps –a**, **vmstat**

Problème : **Espace de pagination quasi-saturé**

Réponse : L'espace de pagination du système doit sans doute être agrandi, à moins que le problème ne soit dû à un processus à perte de mémoire – auquel cas ce processus doit être identifié et l'application corrigée.

Comm. utiles : **ps aucg** (pour examiner l'activité des processus).

smit (pour modifier les caractéristiques de l'espace de pagination).

Problème : **Problème possible au niveau des paramètres de contrôle de charge**

Réponse : Les paramètres de contrôle de charge sont évalués en fonction de l'activité de pagination courante. Par exemple, si un emballement se produit et que le contrôle de charge est désactivé, il convient sans doute de l'activer.

Comm. utiles : **schedtune**

Problème : **Trames de mémoire défectueuses détectées par VMM**

Réponse : Il est peut-être nécessaire d'analyser la mémoire. Comparez la quantité de mémoire installée avec celle effectivement accessible ; si cette dernière est moindre, de la mémoire défectueuse a été identifiée.

Vous pouvez utiliser **/usr/sbin/perf/diag_tool/getvmparms** et consulter la valeur de `numframes` pour déterminer le nombre réel de trames mémoire de 4 ko.

Comm. utiles : **lscfg** | **grep mem** (pour connaître la taille (Mo) de la mémoire installée).

Problème : **Hôte de .nodes inaccessible**

Réponse : Déterminez si le problème provient de l'hôte courant (le fichier **/etc/hosts** a-t-il été modifié ?), de l'hôte distant (est-il en panne ?) ou du réseau (le serveur de noms est-il en panne ?).

Comm. utiles : **ping**

Problème : **Déséquilibre dans la configuration E/S (nombre de disques par carte)**

Réponse : Envisagez de déplacer les disques de façon à ne pas surcharger une seule carte SCSI.

Comm. utiles : **lscfg** (pour étudier la configuration actuelle).

iostat (pour déterminer si la charge actuelle des cartes est déséquilibrée).

Problème : **Déséquilibre dans l'affectation de l'espace de pagination sur les volumes physiques concernés**

Réponse : Envisagez d'unifier la taille des espaces de pagination, sauf pour quelques méga-octets (disons 4) sur l'espace principal (**hd6**).
Un déséquilibre important entre les tailles de ces espaces peut diminuer les performances.

Comm. utiles : **smit**

Problème : **Fragmentation d'un espace de pagination dans un groupe de volumes**

Réponse : Les performances au niveau de la pagination sont meilleures si les zones de pagination sont contiguës sur un volume physique.
Si toutefois les zones sont agrandies, il est possible de créer des fragments éclatés sur tout le disque.

Comm. utiles : **lspv –p hdisk***n* pour chaque volume physique du groupe de volumes. Recherchez plusieurs PP Range dotés du même LVNAME et du même TYPE de "pagination".

Problème : **Déséquilibre significatif de la charge des E/S sur les volumes physiques**

Réponse : Si un volume physique semble peu actif au niveau des E/S, envisagez de déplacer les données des volumes les plus occupés vers les volumes les moins occupés. D'une façon générale, plus les E/S sont uniformément réparties, meilleures sont les performances.

Comm. utiles : **iostat –d 2 20** (pour étudier la répartition actuelle des E/S sur les volumes physiques).

Problème : **Nouveau processus identifié comme gros consommateur de mémoire ou de CPU**

Réponse : Les plus gros consommateurs de CPU et de mémoire sont régulièrement identifiés par PDT. Si un de ces processus n'a pas été détecté auparavant, il est signalé dans un rapport d'incident. Il convient d'examiner ces processus pour déceler d'éventuels dysfonctionnements. PDT ne prend en compte que l'ID de processus. Si un utilisateur gros consommateur connu s'arrête, puis est relancé (avec un id processus différent), il sera identifié comme NOUVEL utilisateur gourmand.

Comm. utiles : **ps aucg** (pour examiner tous les processus et leur activité).

Problème : **Fichier de .files présentant une croissance (ou un déclin) de taille systématique**

Réponse : Examinez la taille actuelle. Considérez le taux de croissance prévisible. Quel utilisateur (ou application) génère les données ? Par exemple, le fichier **/var/adm/wtmp** est susceptible de grandir inconsidérément. S'il devient trop grand, les délais de connexion peuvent s'allonger. Parfois, la solution est de supprimer le fichier. Mais l'essentiel dans tous les cas est d'identifier l'utilisateur à l'origine de cette croissance et d'étudier avec lui le moyen de résoudre le problème.

Comm. utiles : **ls –al** (pour examiner la taille des fichiers/répertoires).

Problème : **Système de fichiers ou espace de pagination présentant une croissance (ou un déclin) systématique au niveau de l'utilisation de l'espace**

Réponse : Evaluez le taux de croissance et le délai de saturation prévus. Il peut être nécessaire d'agrandir le système de fichiers (ou l'espace de pagination). Mais cette opération peut induire des effets pervers (processus à perte de mémoire, par exemple).

Comm. utiles : **smit** (pour manipuler systèmes de fichiers/espaces de pagination)

ps aucg, **svmon** (pour étudier l'activité des processus en mémoire virtuelle).

filemon (pour étudier l'activité du système de fichiers).

Problème : **Dégradation du temps de réponse ping ou du pourcentage de paquets perdus sur un hôte de .nodes**

Réponse : L'hôte en question est-il confronté à des problèmes de performances ? Le réseau est-il confronté à ce type de problème ?

Comm. utiles : **ping**, **rlogin**, **rsh** (pour chronométrer les charges de travail connues sur l'hôte distant)

Problème : **Un processus getty consomme trop de temps CPU**

Réponse : Les processus getty utilisant plus qu'un faible pourcentage de CPU peuvent être en erreur. Dans certaines situations, ces processus peuvent consommer du CPU système alors qu'aucun utilisateur n'est effectivement connecté. La solution est en général de mettre fin au processus.

Comm. utiles : **ps aucg** (pour déterminer le pourcentage de CPU utilisé).

Problème : **Processus gros consommateur de CPU ou de mémoire présentant une croissance (ou un déclin) systématique au niveau de l'utilisation des ressources**

Réponse : Les gros consommateurs connus de CPU et de mémoire sont constamment surveillés pour déceler si leur demande augmente. En tant que gros consommateurs, une croissance linéaire de leur demande doit retenir l'attention à plusieurs niveaux. Si cette croissance est normale, elle figure parmi les éléments essentiels de planification de la capacité. Si elle est excessive, il convient d'envisager de modifier la charge (ou de résoudre un problème chronique, telle une insuffisance de mémoire).

Comm. utiles : **ps aucg**

Problème : **maxuproc signalé probablement insuffisant pour un userid donné**

Réponse : Il est problable que cet utilisateur approche du seuil de **maxuproc**.

maxuproc est un paramètre système qui limite le nombre de processus que les utilisateurs non racine sont autorisés à maintenir actifs simultanément. Si la limite est trop basse, le travail de l'utilisateur peut être différé ou arrêté. Mais un utilisateur peut également, par accident, créer plus de processus que nécessaire. Un supplément d'investigation s'impose dans les deux cas : l'utilisateur doit être consulté pour déterminer précisément ce qui se passe.

Comm. utiles : **lsattr –E –l sys0 | grep maxuproc**

Pour déterminer la valeur actuelle de **maxuproc** (bien qu'elle soit également indiquée directement dans le message PDT).

chdev –l sys0 –a maxuproc=100

Pour passer à 100 la valeur de **maxuproc** (par exemple). Vous devez être utilisateur racine.

Problème : **Un indicateur WORKLOAD TRACKING indique une tendance ascendante**

Réponse : La réponse dépend de l'indicateur concerné :

loadavg – charge moyenne sur 15 minutes

Le niveau de conflits sur le système est en hausse. Examinez le reste du rapport PDT à la recherche d'indicateurs de goulots d'étranglement système (par exemple, une utilisation conséquente d'espace de pagination peut révéler une insuffisance de mémoire, un déséquilibre des E/S un incident au niveau du sous-système d'E/S, etc.).

nusers – nombre total d'utilisateurs connectés

Les utilisateurs sont en nombre croissant sur le système. Cette information est capitale pour planifier la capacité. Cette croissance est-elle prévue ? Peut-elle être expliquée ?

nprocesses – nombre total de processus

Les processus sur le système sont en nombre croissant. Les utilisateurs butent-ils sur le seuil `maxuproc` ? Peut-être y a-t-il des applications qui s'emballent, générant trop de processus en parallèle.

STAT_A – nombre de processus actifs

Une évolution indique que le délai d'attente de CPU par les processus est de plus en plus long.

STAT_W – nombre de processus permutés

Une évolution indique que les conflits de processus pour l'accès à la mémoire sont excessifs.

STAT_Z – nombre de processus zombis

L'existence des zombis doit être éphémère. Si leur nombre augmente sur un système, vous devez étudier le problème.

STAT_I – nombre de processus libres

Cette indication n'est pas forcément signe de problème.

STAT_T – nombre de processus arrêtés après réception d'un signal

Une évolution peut indiquer une erreur de programmation.

STAT_*x* – (*x* étant un caractère valide quelconque de la sortie de la commande **ps**, indiquant un état de processus non répertorié ci-dessus)

L'interprétation d'une évolution dépend ici de la signification de *x*.

cp – délai requis pour la copie d'un fichier de 40 ko

Un accroissement de ce délai indique à l'évidence une dégradation du sous-système d'E/S.

idle_pct_cpu0 – pourcentage libre pour le processeur 0

Un accroissement peut indiquer une recrudescence des conflits au niveau des ressources autres que CPU (pagination ou E/S, par exemple). Cette tendance mérite attention car elle indique que les ressources CPU ne sont pas correctement exploitées.

idle_pct_avg – pourcentage libre moyen pour tous les processeurs

Un accroissement peut indiquer une recrudescence des conflits au niveau des ressources autres que CPU (pagination ou E/S, par exemple). Cette tendance mérite attention car elle indique que les ressources CPU ne sont pas correctement exploitées.

Boîte à outils PTX

La boîte à outils PTX (Performance Toolbox for AIX) est un logiciel sous licence qui permet d'afficher graphiquement diverses mesures de performances. L'affichage graphique facilite grandement la lecture des performances par rapport aux listings de chiffres proposés par les programmes ASCII équivalents. PTX permet également de combiner les informations issues de différentes commandes de contrôle de performances AIX.

PTX est décrit en détail dans *Performance Toolbox 1.2 and 2.1 for AIX: User's Guide*.

Diagnostics en fonction d'un symptôme particulier

Face à une anomalie de performance, l'analyste pourra déterminer plus aisément les causes du problème s'il en connaît la nature.

Cette section traite des points suivants :

- "Ralentissement d'un programme particulier"
- "Ralentissement général à un moment donné"
- "Ralentissement général et imprévisible"
- "Ralentissement des programmes d'un utilisateur particulier"
- "Ralentissement simultané de certains systèmes du réseau local"

En cas de ralentissement général intermittent sur une unité ou un service en particulier, reportez-vous, selon le cas, à :

- "Contrôle et optimisation des E/S disque"
- "Contrôle et optimisation de la mémoire"
- "Contrôle et optimisation de la CPU"
- "Contrôle et optimisation des E/S de communications."

Ralentissement d'un programme particulier

Ce cas, apparemment banal, doit cependant être clairement analysé :

- Le programme a-t-il toujours fonctionné lentement ?

  S'il s'agit d'un ralentissement récent, vous devez déterminez quel changement peut en être à l'origine.

- Le code source a-t-il été modifié ou une nouvelle version installée ?

  Dans l'affirmative, contactez le programmeur ou le fournisseur.

- L'environnement a-t-il été modifié ?

  Si un fichier utilisé par le programme (y compris son propre exécutable) a été déplacé, il se peut que le programme subisse les retards liés au réseau local ou que des fichiers sollicitent simultanément un mécanisme d'accès sur un seul disque et non sur plusieurs comme auparavant.

  Si l'administrateur système a modifié les paramètres d'optimisation du système, le programme peut subir des contraintes qu'il n'avait pas auparavant. Par exemple, si le mode de calcul des priorités a été modifié via la commande **schedtune –r**, les programmes auparavant rapides en arrière-plan peuvent être ralentis et ceux en avant-plan accélérés.

- Le programme est-il écrit en **awk**, **csh** ou dans un autre langage interprétatif ?

  Les langages interprétatifs, utiles pour écrire rapidement un programme, ne sont pas optimisés par un compilateur. Par ailleurs, avec un langage de type **awk**, quelques caractères suffisent parfois pour demander l'exécution d'opérations extrêmement lourdes en E/S ou en calcul. Il est souvent utile de vérifier ces programmes pas à pas ou de les faire réviser de façon informelle pour mettre en évidence le nombre d'itérations qu'implique chaque opération.

- Le programme est-il toujours aussi lent ?

  Une partie de la mémoire est utilisée par le système de fichiers d'AIX pour conserver des pages de fichiers en vue d'un usage ultérieur. Si un programme limité en disque est exécuté deux fois de suite de façon très rapprochée, il doit normalement être plus rapide la seconde fois que la première. Un phénomène similaire peut être observé avec des

programme exploitant NFS et DFS. Il en est de même avec de gros programmes, tels que des compilateurs. Même si l'algorithme du programme n'est pas limité en disque, la durée de chargement d'un exécutable volumineux sera plus élevée à la première exécution qu'aux suivantes.

- Si le programme a toujours été très lent ou qu'aucune modification d'environnement ne peut justifier son ralentissement, il faut étudier les ressources dont il dépend.

  Reportez-vous à "Identification des ressources limitatives", page 12-23 pour connaître les techniques de localisation des goulots d'étranglement.

Ralentissement général à un moment donné

Le ralentissement aux heures de pointe est un phénomène bien connu dû à l'arrivée massive et habituelle des utilisateurs d'une entreprise sur un système. Mais il ne s'agit pas toujours simplement d'un problème de concentration de la charge de travail. Parfois, ce phénomène peut trahir un déséquilibre qui ne se manifeste pour l'instant que lors des charges lourdes. D'autres sources de ralentissement périodique doivent également être envisagées.

- Si vous exécutez **iostat** et **netstat** sur une période couvrant la durée du ralentissement (ou que vous avez précédemment extrait des données du dispositif de contrôle), avez-vous constaté une utilisation plus intensive de certains disques ? Le pourcentage de CPU libre est-il régulièrement proche de zéro ? Le nombre de paquets envoyés ou reçus est-il anormalement élevé ?

  Si un déséquilibre apparaît au niveau des disques, reportez-vous à "Contrôle et optimisation des E/S disque", page 8-1.

  Si la CPU est saturée, lancez **ps** pour identifier les programmes exécutés durant cette période. Le script présenté à "Commandes de contrôle iostat, netstat et vmstat" vous aidera à trouver les sources qui monopolisent la CPU.

  Si le ralentissement semble aberrant (paralysie aux heures de repas, par exemple), assurez-vous qu'il ne s'agit pas d'un programme "pathologique" (tel Graphic Xlock ou un jeu informatique). Certaines versions de Xlock font un usage "boulimique" de temps CPU pour afficher des graphiques sur un écran vide. Il se peut également qu'un utilisateur profite de ce moment opportun pour lancer un programme très gourmand en CPU.

- A moins que votre fichier **/var/adm/cron/cron.allow** ne soit vide, il n'est pas inutile de vérifier si le répertoire **/var/adm/cron/crontab** contient des opérations très lourdes. Par exemple, il se peut que des utilisateurs aient demandé la sauvegarde de tous les fichiers de leur répertoire personnel, toutes les heures, dans un répertoire monté sur NFS.

S'il s'avère que le problème provient d'un conflit entre les activités d'avant-plan et des programmes longs, très consommateurs en CPU, exécutés (ou appelés à l'être) en arrière-plan, lancez **schedtune –r –d** pour donner priorité aux activités d'avant-plan. Reportez-vous à "Optimisation du calcul de la priorité des processus avec schedtune", page 6-23.

Ralentissement général et imprévisible

L'outil le mieux adapté à ce problème est un détecteur de surcharge, tel le programme **filtd** de **xmperf** (composant de PTX). Ce programme peut être configuré pour exécuter les scripts shell ou recueillir des informations lorsque certaines conditions sont réunies. Vous pouvez créer votre propre outil selon le même modèle à l'aide de scripts shell contenant les commandes **vmstat**, **netstat** et **ps**.

Si l'incident ne se produit que sur un seul système d'un environnement distribué, il s'agit sans doute d'un programme "pathologique" ou de deux qui interagissent de façon aléatoire.

Ralentissement des programmes d'un utilisateur particulier

Parfois, un système semble avoir décidé de "harceler" un utilisateur particulier.

- Dans ce cas, vous devez tenter de quantifier le problème : demandez à l'utilisateur quelles sont les commandes qu'il a l'habitude d'utiliser et exécutez-les en association avec la commande **time**, comme suit :

```
$ time cp .profile testjunk
real    0m0.08s
user    0m0.00s
sys     0m0.01s
```

  Puis exécutez-les sous un ID utilisateur valide. Le temps réel indiqué sur la ligne `real` est-il différent ?

- D'une exécution à l'autre, un programme consomme normalement à peu près le même temps CPU (`user+sys`), mais il se peut parfois que le temps réel (`real`) utilisé soit différent du fait des opérations E/S, plus nombreuses ou plus lentes. Les fichiers de l'utilisateur sont-il implantés sur un répertoire monté sur NFS ? Ou sur un disque intensément utilisé pour d'autres raisons ?
- Vérifiez le fichier `.profile` de l'utilisateur : les spécifications `$PATH` sont peut-être étranges. Par exemple, si vous recherchez sans succès une paire de répertoires montés sur NFS avant d'explorer `/usr/bin`, tout sera plus long.

Ralentissement simultané de certains systèmes du réseau local

Passer de systèmes autonomes à des systèmes distribués se fait rarement sans difficultés : le peu de temps imparti à reconfigurer les systèmes et la méconnaissance du coût de certaines fonctions en sont souvent la cause. Par ailleurs, en plus de l'optimisation de la configuration LAN des MTU et mbuf (voir le chapitre Contrôle et optimisation des E/S de communications), vous devez rechercher les pathologies propres au réseau local et les lourdeurs résultant de la combinaison de mesures individuelles, justifiées en elles-mêmes.

- Certains défauts de logiciels ou de micrologiciels peuvent saturer sporadiquement le réseau local de paquets de diffusion ou autres.

  Lorsque le réseau est inondé de paquets de diffusion, même les systèmes qui le sollicitent peu peuvent être ralentis par les interruptions incessantes et le temps CPU consommé par la réception et le traitement des paquets. Les dispositifs d'analyse des réseaux locaux sont plus à même de détecter et localiser ces défaillances que les outils de performances AIX (PTX).

- Deux réseaux locaux sont-ils connectés via un système AIX ?

  L'utilisation d'un système AIX comme routeur monopolise beaucoup de temps CPU pour le traitement et la copie des paquets. Des interférences avec d'autres travaux en cours sur le système AIX peuvent se produire. La connexion à des réseaux locaux via des ponts et des routeurs matériels dédiés est généralement plus rentable et durable.

- Tous les montages NFS sont-ils justifiés ?

  A un certain stade de développement des configurations distribuées, les montages NFS sont utilisés pour donner aux utilisateurs des nouveaux systèmes accès à leur répertoire personnel sur le système d'origine. Cette opération simplifie la transition initiale mais impose un coût de communication de données continue. Il arrive que des utilisateurs d'un système A utilisent essentiellement les données d'un système B et vice versa.

  L'accès à des fichiers via NFS représente un coût considérable en terme de trafic LAN, temps CPU serveur et client et temps de réponse pour l'utilisateur final. En principe, il faut autant que possible que les utilisateurs et les données résident sur le même système. Seule exception à cette règle, les cas où un impératif justifie le coût et le temps supplémentaire pour des données distantes : par exemple, la nécessité de centraliser

les données pour assurer un contrôle et une sauvegarde fiable ou encore de s'assurer que tous les utilisateurs travaillent sur la dernière version d'un programme.

Si ces impératifs exigent un taux important d'échanges client/serveur NFS, il est préférable de dédier un système au rôle de serveur plutôt que de travailler avec plusieurs systèmes mi-serveurs, mi-clients.

- Les programmes ont-ils été correctement portés pour utiliser les appels de procédure à distance RPC (et cela se justifiait-il) ?

  Le plus simple pour porter un programme sur un environnement distribué est de remplacer les appels de programmes par des RPC sur une base 1:1. Mais il faut savoir que l'écart de performances entre des appels de programme locaux et des appels RPC est encore plus élevé que l'écart entre les E/S disque locales et les E/S NFS. Dès lors que les RPC se révèlent indispensables, il est préférable de les traiter par lots chaque fois que possible.

Ralentissement général intermittent sur une unité ou un service

Assurez-vous que vous avez suivi les recommandations de configuration du guide relatif au sous-système en cause et/ou au chapitre "Contrôle et optimisation" concerné dans le présent manuel.

Diagnostics à l'aide de PerfPMR

La fonction du module PerfPMR est de compléter les comptes rendus d'anomalies de performances sous AIX, jusqu'à ce qu'ils contiennent suffisamment de données pour que BULL puisse établir un diagnostic. De ce fait, les scripts shell de PermPMR peuvent être utiles à d'autres analystes. PerfPMR est un module en option du BOS (Base Operating System) d'AIX version 4.1. Il se trouve dans /**usr**/**sbin**/**perf**/**pmr**. Reportez-vous à "Installation de PerfPMR d'AIX version 4.1". Une version de PerfPMR pour AIX version 3.2.5 est également disponible (reportez-vous à "Obtention et installation de PerfPMR (AIX version 3.2.5)").

perfpmr est le script de plus haut niveau du module, mais les données qu'il collecte (telles que des informations de configuration) sont généralement déjà connues de l'analyste local. Le script de plus bas niveau, **monitor**, se charge de recueillir un ensemble d'informations coordonnées pendant un intervalle donné (en secondes) et de résumer ces données. La syntaxe de **monitor** est :

monitor *secondes* [**–n**] [**–p**]

Le paramètre *secondes* doit être supérieur ou égal à 60. Si *secondes* est inférieur ou égal à 600, le compte rendu périodique est généré toutes les 10 secondes ; sinon, l'intervalle est de 60 secondes. L'indicateur **–n** élimine la collecte des données **netstat** et **nfsstat**. L'indicateur **–p** élimine la collecte des données du profil du processus (voir ci-dessous). Il est déconseillé d'exécuter le script **monitor** en même temps qu'une opération qui sollicite l'utilitaire de suivi du système.

Chaque requête **monitor** crée :

- Un fichier **monitor.int** contenant le résultat :
  - des commandes **ps –elk** et **ps gv** exécutées au début et à la fin de la période de contrôle (sorties combinées),
  - d'une commande **sar –A** avec l'intervalle approprié,
  - d'une commande **iostat** avec l'intervalle approprié. Le rapport initial, cumulé, est omis.
  - d'une commande **vmstat** avec l'intervalle approprié. Le rapport initial, cumulé, est omis.
- Un fichier **monitor.sum** contenant :
  - pour les processus actifs en début et fin du contrôle, les écarts entre les valeurs initiales et finales des statistiques d'utilisation des ressources,
  - les lignes "`Average`" issues de la sortie de la commande **sar –A**,
  - la moyenne des statistiques de l'intervalle **iostat**,
  - la moyenne des statistiques de l'intervalle **vmstat**,
  - l'écart entre les statistiques avant/après générées par la commande **vmstat –s**.
- Si l'option **–n** *n'est pas* spécifiée, un fichier **netstat.int** contenant :
  - la sortie en début de contrôle des commandes :

    netstat –v

    netstat –m

    netstat –rs

    netstat –s
  - la sortie d'une commande **netstat** avec l'intervalle approprié,
  - la sortie en fin de contrôle des commandes :

netstat –v

netstat –m

netstat –rs

netstat –s

- Si l'option **–n** *n'est pas* spécifiée, un fichier **nfsstat.int** contenant :
  - la sortie en début et fin de contrôle d'une commande **nfsstat –csnr**.
- Si l'option **–p** *n'est pas* spécifiée, deux fichiers, **Pprof.stt** et **Pprof.flow. Pprof.stt**, contiennent les heures de début et de fin de l'exécution. **Pprof.flow** contient les données du profil du processus. Les colonnes du fichier **Pprof.flow** sont :
  a. Nom du processus
  b. ID processus
  c. Heure de la première occurrence du processus dans l'intervalle de mesure
  d. Heure de la dernière occurrence du processus
  e. Temps d'exécution total du processus
  f. Indicateurs Begin/End (la somme des deux est donnée plus loin). Indique l'état du processus en début et fin d'exécution :

```
Begin:
  execed:           0
  forked:           1
  Alive at Start:   2
End:
  Alive at end:     0
  execed away:      4
  Exited:           8
```

  g. ID processus parent

Contrôle avant modification

Une des fonctions les plus utiles de PerfPMR est de créer une ligne de base de la configuration et des performances avant toute modification logicielle ou matérielle significative. Comme vous le faites pour les fichiers critiques, vous devez sauvegarder les configurations et leurs performances : si vous constatez une dégradation des performances, vous disposez alors de toutes les données utiles pour analyser précisément la situation avant et après modification.

Pour obtenir les données les plus complètes possibles, exécutez :

```
$ perfpmr 3600
```

au cours de l'heure la plus chargée de la journée. Les fichiers résultants de cette mesure se trouvent dans le répertoire **/var/perf/tmp**. (Si vous exploitez un système antérieur à la version 4, les fichiers résultants se trouvent dans le répertoire de travail courant.) Veillez à placer ces fichiers dans un endroit sûr avant toute modification.

Identification des ressources limitatives

Cette section traite des points suivants :

- ”Aperçu des performances du système”
- ”Détermination du facteur limitatif sur un programme”
- ”Disque ou mémoire”

Aperçu des performances du système

Le meilleur outil pour avoir une vue d’ensemble de l’utilisation des ressources dans un environnement multi-utilisateur est sans doute la commande **vmstat**. La commande **vmstat** rend compte de l’activité de la CPU et des E/S disque, ainsi que les données d’utilisation de la mémoire. La commande

```
$ vmstat 5
```

commande à **vmstat** d’écrire une ligne de rapport sur l’activité du système toutes les 5 secondes. Aucun compte n’ayant été spécifié à la suite de cet intervalle, la génération des rapports se poursuit jusqu’à annulation de la commande.

Le relevé **vmstat** ci-dessous concerne un système exécutant AIXwindows et plusieurs applications associées (certains intervalles de faible activité ont été supprimés) :

```
procs     memory             page                   faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b    avm    fre  re  pi  po  fr   sr  cy  in    sy  cs us sy id wa
 0  0   8793     81   0   0   0   1    7   0 125    42  30  1  2 95  2
 0  0   8793     80   0   0   0   0    0   0 155   113  79 14  8 78  0
 0  0   8793     57   0   3   0   0    0   0 178    28  69  1 12 81  6
 0  0   9192     66   0   0  16  81  167   0 151    32  34  1  6 77 16
 0  0   9193     65   0   0   0   0    0   0 117    29  26  1  3 96  0
 0  0   9193     65   0   0   0   0    0   0 120    30  31  1  3 95  0
 0  0   9693     69   0   0  53 100  216   0 168    27  57  1  4 63 33
 0  0   9693     69   0   0   0   0    0   0 134    96  60 12  4 84  0
 0  0  10193     57   0   0   0   0    0   0 124    29  32  1  3 94  2
 0  0  11194     64   0   0  38 201 1080   0 168    29  57  2  8 62 29
 0  0  11194     63   0   0   0   0    0   0 141   111  65 12  7 81  0
 0  0   5480    755   3   1   0   0    0   0 154   107  71 13  8 78  2
 0  0   5467   5747   0   3   0   0    0   0 167    39  68  1 16 79  5
 0  1   4797   5821   0  21   0   0    0   0 191   192 125 20  5 42 33
 0  1   3778   6119   0  24   0   0    0   0 188   170  98  5  8 41 46
 0  0   3751   6139   0   0   0   0    0   0 145    24  54  1 10 89  0
```

Les colonnes pertinentes pour l’évaluation initiale sont les colonnes `pi` et `po` dans `page` et les quatre colonnes de la rubrique `cpu`.

- Les entrées `pi` et `po` correspondent respectivement aux chargements et aux déchargements de page dans l’espace de pagination. Si une E/S dans l’espace de pagination se produit, la charge de travail approche (voire dépasse) les limites de mémoire du système.
- Si la somme des taux d’utilisation de CPU `us` et `sy` (utilisateur et système) excède 80 % dans un intervalle donné de 5 secondes, la charge de travail s’approche des limites CPU du système pendant cet intervalle.
- Un pourcentage d’attente des E/S (`wa`) non nul (colonnes `pi` et `po` à zéro) signifie qu’un certain temps d’attente est marqué sur les E/S de fichiers non coordonnées et qu’une partie de la charge de travail est limitée en E/S.

S’approcher des limites signifie qu’une partie de la charge subit un ralentissement du fait d’une ressource critique. L’allongement des temps de réponse n’est peut-être pas subjectivement significatif, mais tout supplément de charge à ce niveau entraînera une détérioration rapide des performances.

Si **vmstat** signale un temps d'attente des E/S important, exécutez **iostat** pour disposer d'informations plus détaillées. La commande

```
$ iostat 5 3
```

commande à **iostat** d'établir toutes les 5 secondes un bref relevé des E/S et de l'activité de la CPU. Le chiffre 3 final indique de générer trois relevés.

Le relevé **iostat** ci-dessous concerne un système doté de la même charge de travail que dans l'exemple précédent (**vmstat**), mais à un autre moment. Le premier relevé fournit le cumul d'activité depuis le dernier amorçage, et les suivants, l'activité au cours de l'intervalle de 5 secondes qui précède :

```
tty:    tin     tout   cpu:   % user   % sys   % idle   % iowait
        0.0      4.3            0.2      0.6      98.8      0.4

Disks:        % tm_act     Kbps      tps    msps   Kb_read   Kb_wrtn
hdisk0           0.0      0.2       0.0              7993      4408
hdisk1           0.0      0.0       0.0              2179      1692
hdisk2           0.4      1.5       0.3             67548     59151
cd0              0.0      0.0       0.0                 0         0

tty:    tin      tout     cpu:   % user    % sys     % idle  %iowait
        0.0      30.3              8.8      7.2       83.9     0.2

Disks:        % tm_act     Kbps      tps    msps   Kb_read   Kb_wrtn
hdisk0           0.2      0.8       0.2                 4         0
hdisk1           0.0      0.0       0.0                 0         0
hdisk2           0.0      0.0       0.0                 0         0
cd0              0.0      0.0       0.0                 0         0

tty:   tin      tout      cpu:   % user   % sys   % idle   % iowait
       0.0       8.4              0.2      5.8      0.0      93.8

Disks:        % tm_act     Kbps      tps    msps   Kb_read   Kb_wrtn
hdisk0           0.0      0.0       0.0                 0         0
hdisk1           0.0      0.0       0.0                 0         0
hdisk2          98.4    575.6      61.9               396      2488
cd0              0.0      0.0       0.0                 0         0
```

Le premier relevé révèle un déséquilibre au niveau des E/S : la plupart (86,9 % de kilooctets en lecture contre 90,7 % en écriture) concernent le disque `hdisk2` qui contient le système d'exploitation et l'espace de pagination. L'utilisation cumulée de la CPU depuis le dernier amorçage n'est généralement significative que sur un système exploité 24 h sur 24.

La second relevé indique un faible volume de lecture sur le disque `hdisk0`, qui contient un système de fichiers distinct, réservé à l'utilisateur principal du système. L'activité CPU provient de deux programmes d'application et de la commande **iostat** elle-même. La sortie de **iostat**, bien que non volumineuse, est réacheminée vers un fichier, le système n'étant pas suffisamment limité en mémoire pour forcer une sortie pendant cet intervalle.

Dans le troisième relevé, des conditions proches de l'emballement ont été créées, par l'exécution d'un programme qui affecte et remplit un volume important de mémoire (ici, environ 26 Mo). `hdisk2` est actif 98,4 % du temps, ce qui entraîne une attente E/S de 93,8 %. Le fait qu'un seul programme monopolise plus des trois-quarts de la mémoire système (32 Mo) et provoque l'emballement du système met en évidence les limites du contrôle de charge mémoire VMM, page 2-10. Même dans le cas d'une charge plus homogène, la capacité mémoire requise pour chaque élément doit être bien analysée.

Si **vmstat** indique qu'une quantité importante de temps CPU est disponible alors que le système donne des signes de ralentissement, il se peut que vous soyez confronté à des retards causés par des conflits de verrouillage dans le noyau. Sous AIX version 4.1, ce phénomène peut être étudié à l'aide de la commande **lockstat** si Performance Toolbox est installé sur votre système.

Détermination du facteur limitatif sur un programme

Si vous êtes l'unique utilisateur d'un système, vous pouvez, via la commande **time**, déterminer si un programme est ou non dépendant des E/S et de l'utilisation de la CPU :

```
$ time cp foo.in foo.out

real    0m0.13s
user    0m0.01s
sys     0m0.02s
```

Remarque : Les exemples de commande **time** proposés dans ce manuel utilisent la version intégrée au shell Korn. La commande **time** officielle (/**usr**/**bin**/**time)** génère, entre autres inconvénients, un relevé moins détaillé.

Dans cet exemple, le fait que le temps réellement écoulé pour l'exécution de **cp** (0,13 seconde) soit bien supérieur à la somme (0,03 seconde) des temps d'utilisation CPU utilisateur et système, indique que le programme est limité en E/S. Ceci se produit principalement parce que `foo.in` n'a pas fait l'objet d'une lecture récente. La même commande, exécutée sur le même fichier quelques secondes plus tard, indique :

```
real    0m0.06s
user    0m0.01s
sys     0m0.03s
```

La plupart (voire la totalité) des pages de `foo.in` sont encore en mémoire car aucun processus n'est intervenu pour les réclamer et, par ailleurs, la taille du fichier est minime, comparée à la capacité RAM du système. Un petit fichier `foo.out` serait également placé en mémoire et un programme qui l'utiliserait en entrée serait peu dépendant du disque.

Pour déterminer la dépendance d'un programme vis-à-vis du disque, vérifiez l'état de données en entrée : si le programme doit normalement être exécuté sur un fichier qui n'a pas été récemment utilisé, assurez-vous que le fichier test ne se trouve pas en mémoire. S'il s'agit, à l'inverse, d'un programme normalement exécuté dans une séquence standard, et qu'il prend en entrée la sortie du programme précédent, il vous faut *préparer* la mémoire pour garantir l'exactitude de la mesure. Par exemple :

```
$ cp foo.in /dev/null
```

prépare la mémoire en lui intégrant les pages de `foo.in`.

La situation est plus complexe si la taille du fichier est grande par rapport à la capacité RAM. Si la sortie d'un programme est reprise en entrée par le suivant et que la totalité du fichier ne loge pas dans la RAM, le second programme terminera la lecture des pages en début de fichier, déplaçant les pages à la fin. Ces conditions, très difficiles à reproduire artificiellement, équivalent grosso modo à une situation dans laquelle aucune mise en cache n'a lieu sur le disque.

Le cas particulier d'un fichier de taille supérieure à celle de la RAM est traité dans la section suivante.

Disque ou mémoire ?

De même qu'une part importante de la mémoire réelle est réservée à la mise en tampon des fichiers, l'espace de pagination du système est utilisé comme mémoire temporaire pour les données actives d'un programme qui ont été expulsées de la RAM. Supposons qu'un programme lise peu ou pas de données et présente néanmoins des symptômes de dépendance vis–à–vis des E/S. Pire, le ratio de temps réel par rapport au temps utilisateur + système ne s'améliore pas au cours des exécutions successives. On peut en conclure que ce programme est probablement limité en mémoire et que ses E/S sont destinées à l'espace de pagination (d'où elles proviennent probablement). Pour le vérifier, vous pouvez utiliser le script shell `vmstatit` ci-dessous. Le script shell `vmstatit` résume le volumineux compte rendu généré par **vmstat –s**, qui indique les décomptes cumulés de certaines activités système depuis son lancement :

```
vmstat -s >temp.file   # cumulative counts before the command
time $1                # command under test
vmstat -s >>temp.file  # cumulative counts after execution
grep "pagi.*ins" temp.file >>results   # extract only the data
grep "pagi.*outs" temp.file >>results  # of interest
```

Si le script shell est exécuté comme suit :

```
$ vmstatit "cp file1 file2"  2>results
```

le résultat donné dans `results` est :

```
real    0m0.03s
user    0m0.01s
sys     0m0.02s
     2323 paging space page ins
     2323 paging space page ins
     4850 paging space page outs
     4850 paging space page outs
```

Le fait que les données statistiques de pagination avant/après soient identiques nous conforte dans l'idée que la commande **cp** n'est pas limitée en pagination. Une variante plus complète du script shell `vmstatit` peut être utilisée pour refléter la situation réelle :

```
vmstat -s >temp.file
time $1
vmstat -s >>temp.file
echo "Ordinary Input:"               >>results
grep "^[ 0-9]*page ins"    temp.file >>results
echo "Ordinary Output:"              >>results
grep "^[ 0-9]*page outs"   temp.file >>results
echo "True Paging Output:"           >>results
grep "pagi.*outs"          temp.file >>results
echo "True Paging Input:"            >>results
grep "pagi.*ins"           temp.file >>results
```

Toutes les E/S ordinaires effectuées sous AIX étant traitées via VMM, la commande **vmstat –s** rend compte des E/S ordinaires de programmes en termes de chargement/ déchargement de pages. La version ci-dessus du script shell `vmstatit` appliquée à la commande **cp** sur un gros fichier qui n'a pas été récemment utilisé, génère :

```
real    0m2.09s
user    0m0.03s
sys     0m0.74s
Ordinary Input:
    46416 page ins
    47132 page ins
Ordinary Output:
   146483 page outs
   147012 page outs
True Paging Output:
     4854 paging space page outs
     4854 paging space page outs
True Paging Input:
     2527 paging space page ins
     2527 paging space page ins
```

La sortie de la commande **time** confirme l'existence d'une dépendance vis–à–vis des E/S. L'accroissement des "page ins" indique les E/S requises pour satisfaire la commande **cp**. L'accroissement des "page outs" indique que le fichier est suffisamment volumineux pour forcer l'écriture des pages modifiées (pas nécessairement les siennes) à partir de la mémoire. La constance du nombre cumulé d'E/S dans l'espace de pagination confirme que les structures de données construites par la commande **cp** ne sont pas suffisamment étendues pour surcharger la mémoire de la machine test.

L'ordre des E/S dans cette version du script shell `vmstatit` a son importance. En général, les programmes lisent les entrées d'un fichier puis en écrivent la sortie. Mais dans le

processus de pagination, la page de segment actif qui ne peut être accueillie est d'abord déchargée (écrite à l'extérieur) : elle n'est chargée (lue) que lorsque le programme tente d'y accéder. Sur le système test, le déséquilibre entre le nombre de déchargements (`paging space page outs`) et le nombre presque deux fois moindre de chargements (`paging space page ins`) depuis le dernier amorçage révèle que certains programmes ont stocké en mémoire des données qui n'ont pas été réutilisées jusqu'à la fin du programme. "programmes à CPU limitée", page 4-20 donne davantage d'informations. Reportez-vous également à "Contrôle et optimisation de la mémoire", page 7-1.

L'exemple ci-dessous illustre l'impact de la limitation de mémoire sur les statistiques : une commande donnée est exécutée dans un environnement doté de suffisamment de mémoire (32 Mo), puis la taille du système est réduite via une commande **rmss** (reportez-vous à "Estimation de la mémoire requise via rmss", page 7-6). La séquence de commandes :

```
$ cc -c ed.c
$ vmstatit "cc -c ed.c" 2>results
```

prépare dans un premier temps la mémoire en lui intégrant le fichier source de 7 944 lignes et l'exécutable du compilateur C puis mesure l'activité E/S de la seconde exécution :

```
real    0m7.76s
user    0m7.44s
sys     0m0.15s
Ordinary Input:
    57192 page ins
    57192 page ins
Ordinary Output:
   165516 page outs
   165553 page outs
True Paging Output:
    10846 paging space page outs
    10846 paging space page outs
True Paging Input:
     6409 paging space page ins
     6409 paging space page ins
```

Il n'y a visiblement pas de limitation au niveau des E/S. Même pour lire le code source, aucune E/S n'est requise. Si vous exécutez la commande :

```
# rmss -c 8
```

pour ramener à 8 Mo la taille effective de la machine et que vous répétez la même séquence de commandes, vous obtenez :

```
real    0m9.87s
user    0m7.70s
sys     0m0.18s
Ordinary Input:
    57625 page ins
    57809 page ins
Ordinary Output:
   165811 page outs
   165882 page outs
True Paging Output:
    11010 paging space page outs
    11061 paging space page outs
True Paging Input:
     6623 paging space page ins
     6701 paging space page ins
```

Cette fois, les symptômes d'E/S limitées apparaissent :

- le temps écoulé est supérieur au temps CPU total,
- le nombre d'E/S ordinaires à la nième exécution est considérable.

Ces deux symptômes signalent que le manque de mémoire entrave l'exécution du compilateur.

> **Remarque :** Cet exemple illustre les effets d'une insuffisance de mémoire. Aucun effort particulier n'a été fait pour minimiser l'utilisation de la mémoire par les autres processus, la taille absolue imposée au compilateur pour charger cet environnement n'est donc pas une mesure pertinente.

Pour ne pas être contraint de travailler sur une machine artificiellement réduite en taille jusqu'à la prochaine relance, exécutez :

```
# rmss -r
```

Cette commande restitue au système d'exploitation la mémoire "confisquée" par la commande **rmss** et redonne ainsi au système sa capacité normale.

Gestion de la charge de travail

Si vous avez épuisé toutes les possibilités d'optimisation du système et des programmes sans parvenir à un niveau de performances satisfaisant, il vous reste trois solutions :

- vous contenter de ce niveau de performances,
- mettre à niveau la ressource limitative,
- prendre des mesures de gestion de la charge.

Si vous optez pour la première solution, vous risquez de démotiver vos utilisateurs les moins stoïques à force de frustration. Si vous optez pour la seconde solution, il y aura toujours quelqu'un pour vous demander des comptes sur cette dépense supplémentaire. Et il vous faudra prouver que vous avez épuisé toutes les autres possibilités sur le système courant. Bref, il ne vous reste plus que la troisième solution : la gestion de la charge.

Gérer la charge de travail signifie simplement en analyser les éléments pour évaluer leur degré d'urgence. Tous les travaux n'ont en effet pas la même priorité : ce rapport est utile qu'il soit exécuté à 3 heures du matin ou à 16 heures la veille. La différence est qu'à 3 heures du matin, les cycles CPU qu'il utilise seraient sinon inexploités. Pour exécuter un programme à une heure déterminée ou à intervalles réguliers, vous disposez des commandes **at** et **crontab**.

De même, certains programmes devant être exécutés dans la journée peuvent être assortis d'une priorité réduite : leur exécution prendra plus de temps, mais la compétition avec les processus réellement urgents sera moindre.

Une autre méthode, dérivée de la première, consiste à transférer le travail d'une machine à une autre, pour exécuter, par exemple, une compilation sur la machine où réside le code source. Cet équilibrage de la charge exige davantage de planification et de contrôle : réduire la charge sur le réseau et augmenter la charge CPU sur un serveur est une opération qui, mal maîtrisée, peut se solder par une perte nette.

Chapitre 13. Gestion d'un possible défaut de performance AIX

Si vous pensez avoir détecté un défaut au niveau des performances d'AIX, vous disposez d'outils et de procédures pour faire part du problème et fournir les données permettant de l'analyser. Outils et procédures sont conçus pour vous donner une solution aussi précise et rapide que possible avec un minimum d'effort et de temps de votre part.

Cette rubrique traite des points suivants :

- "Base de mesure"
- "Compte rendu du problème"
- "Obtention et installation de PerfPMR (AIX version 3.2.5)"
- "Installation de PerfPMR à partir du Web (inclut la version 4)"
- "Données d'analyse"

Base de mesure

Les problèmes de performance apparaissent souvent immédiatement après une modification du système, matérielle ou logicielle. A moins qu'une mesure pré-modification n'ait été effectuée, base possible de comparaison des performances "avant/après", quantifier le problème est impossible. Seule la collecte d'informations exhaustives sur la configuration et les performances via le module PerfPMR permet une analyse efficace. Reportez-vous à "Contrôle avant modification".

Disposer de l'outil PDT (Performance Diagnostic Tool) installé et opérationnel est également utile pour étudier l'ensemble des performances du système.

Compte rendu du problème

Vous devez rendre compte des problèmes suspectés au niveau des performances AIX au service BULL compétent. Pour ce faire, passez par votre canal habituel. Si vous n'êtes pas familier de ce type d'opération, consultez votre représentant BULL.

Au moment de ce rapport, vous devez fournir les informations suivantes :

- Une description du problème qui permette d'explorer la base d'incidents pour déterminer si un problème de ce type a déjà été signalé.
- Quel aspect de votre analyse vous a conduit à penser qu'AIX était en cause ?
- Dans quelle configuration matérielle/logicielle avez-vous rencontré ce problème ?
  - Le problème est-il limité à un seul système, ou en affecte-t-il plusieurs ?
  - Quels sont le modèle, la taille mémoire et le nombre et la taille des disques des systèmes en cause ?
  - Quels types de réseaux locaux et de supports de communications sont connectés au(x) système(s) ?
  - La configuration comporte-t-elle des systèmes non-AIX ? Non-UNIX ?
- Quelles sont les caractéristiques du programme ou du travail rencontrant le problème ?

  - Une analyse avec **time**, **iostat** et **vmstat** indique-t-elle une limitation de CPU ou d'E/S ?
  - Les travaux sont-ils exécutés sur : station de travail, serveur, multi-utilisateur ou un panachage de ces types de système ?
- Quels sont les objectifs non atteints ?
  - L'objectif principal est-il défini en termes de temps de réponse du terminal ou de la console, du débit ou des réponses en temps réel ?
  - Les objectifs ont-ils été fixés sur la base de mesures effectuées sur un autre système AIX ? Si oui, quelle en est la configuration ?

Si l'incident est signalé pour la première fois, vous vous verrez attribuer un numéro de référence PMR (à indiquer lors de tout renseignement complémentaire).

Vous serez probablement invité à fournir des données pour aider BULL à analyser le problème. Pour les collecter, BULL fournit un module d'outils, appelé PerfPMR.
Sous AIX version 3.2.5, il s'agit d'un outil informel disponible auprès de votre représentant BULL. Sous AIX version 4.1, PerfPMR est un module installable en option, fourni sur le support du système d'exploitation de base (BOS) AIX.

Obtention et installation de PerfPMR (AIX version 3.2.5)

Procurez-vous une copie de PerfPMR AIX version 3.2.5 auprès de votre représentant BULL.

Pour installer PerfPMR :

1. Connectez-vous en tant qu'utilisateur `root` ou passez par la commande **su** pour en obtenir les droits.
2. Créez le répertoire **perfpmr** et passez-y (cet exemple suppose que le répertoire est créé sous /**tmp**).

   ```
   # cd /tmp
   # mkdir perfpmr
   # cd perfpmr
   ```

3. Copiez le fichier **tar** compressé à partir de la disquette (cet exemple suppose que l'unité de disque utilisée est **fd0**) :

   ```
   # tar -xvf/dev/fd0 perfpmr.tarbinz
   ```

4. Renommez le fichier **tar** :

   ```
   # mv perfpmr.tarbinz perfpmr.tarbin.Z
   ```

5. Décompressez le fichier **tar** :

   ```
   # uncompress perfpmr.tarbin.Z
   ```

6. Récupérez les scripts shell du fichier **tar** :

   ```
   # tar -xvf perfpmr.tarbin
   ```

7. Installez les scripts shell :

   ```
   # ./Install
   ```

Installation de PerfPMR à partir du Web (inclut la version 4)

Pour obtenir la bonne version de PerfPMR, visitez notre site Web à l'adresse **www.ibm.com**. Saisissez `perfpmr` dans la zone de texte Search et appliquez la recherche à l'ensemble du site (`All of IBM`). Le résultat de la recherche apparaît dans une fenêtre ayant pour titre "AIX Performance PMR Data Collection Scripts – perfpmr." Cliquez sur le bouton de téléchargement pour tranférer les fichiers depuis le site FTP contenant PerfPMR. Plusieurs versions de PerfPMR sont répertoriées (perf32, perf41, perf42 et perf43 par exemple).

Chaque dossier contient :

fichier **perf*xx*.tar.Z**	Fichier tar compressé contenant PerfPMR
message	Description du contenu
license.agreement	Accord de licence international BULL
readme	Instructions sur l'obtention et l'installation de PerfPMR, la collecte des données et l'envoi de ces données à BULL

Pour installer PerfPMR à partir du Web :

1. Téléchargez la bonne version de PerfPMR dans le répetoire /**tmp**
2. Connectez-vous en tant qu'utilisateur root ou utilisez la commande **su** pour en obtenir les droits.
3. Créez le répertoire **perf41** (par exemple) et passez–y

```
# mkdir /tmp/perf41

# cd /tmp/perf41
```

4. Récupérez les scripts shell du fichier tar compressé :

```
# zcat /tmp/perf41.tar.Z | tar –xvf –
```

5. Installez les scripts shell :

```
# sh ./Install
```

Le processus d'installation place le module PerfPMR dans le répertoire /**usr**/**sbin**/**perf**/**pmr**. Le module occupe environ 200 ko d'espace disque.

Données d'analyse

Tous les éléments suivants doivent être intégrés aux informations fournies au service de maintenance, lors de la première collecte :

- Un moyen de reproduire le problème
  - Si possible, joignez un programme ou un script shell illustrant le problème
  - A défaut, décrivez précisément les conditions sous lesquelles a eu lieu le problème.
- Les données collectées par les outils PerfPMR
  - Sur chaque système concerné
  - Au même moment
  - Pendant l'occurrence du problème.

- L'application mise en cause par le problème
  - Si l'application est un produit logiciel (ou en dépend), indiquez-en la version et la révision exactes, même s'il ne s'agit pas d'un produit BULL.
  - Si le code source de l'application a été créé par un utilisateur, indiquez l'ensemble complet des paramètres du compilateur ayant servi à créer l'exécutable.

Capture des données

Pour capturer et conditionner les données sous une forme exploitable, exécutez les étapes suivantes sur chacun des systèmes concernés. Si possible, exécutez l'étape 6 simultanément (ou presque) sur tous les systèmes.

1. Connectez-vous en tant qu'utilisateur root ou utilisez la commande **su** pour en obtenir les droits.
2. PerfPMR capture davantage d'informations si les outils **tprof**, **filemon** et **netpmon** sont disponibles. Sous AIX version 4.1, ils sont fournis comme élément de la boîte à outils Performance Toolbox for AIX. Pour déterminer si ces outils ont été installés, entrez :

   ```
   $ lslpp -lI perfagent.tools
   ```

   Si ce module est installé, les outils sont disponibles.
3. Vérifiez que la variable PATH inclut le répertoire contenant les exécutables PerfPMR.

   Sous AIX version 4.1, ajoutez /**usr**/**sbin**/**perf**/**pmr** à PATH. Par exemple :

   ```
   # echo $PATH
   /usr/bin:/etc:/usr/local/bin:/usr/ucb:.:
   # PATH=$PATH:/usr/sbin/perf/pmr:
   # export PATH
   ```

   Sous la version 3, ajoutez à PATH le répertoire sous lequel vous avez installé PerfPMR (à la place de /**usr**/**sbin**/**perf**/**pmr**) et le répertoire des outils de performance, /**usr**/**lpp**/**bosperf**.
4. Sous la version 4, la sortie de **perfpmr** est écrite dans /**var**/**perf**/**tmp**. Sous la version 3, vous devez :
   a. Passer (via **cd**) à un répertoire adéquat, tel /**tmp**, dans un système de fichiers disposant d'au moins 5 Mo d'espace libre.
   b. Créer un sous-répertoire pour les données et y passer :

      ```
      # mkdir perfdata
      # cd perfdata
      ```

5. Surveillez l'activité système pendant 1 heure :

   ```
   # perfpmr 3600
   ```

   (dans la version 3, **perfpmr** s'appelle **perfpmr.sh**.)

6. Fusionnez les fichiers en un seul fichier **tar** compressé :

   ```
   # cd ..
   # tar -cvf numéro-pmr.tarbin.perfdata
   # compress numéro-pmr.tarbin
   ```

   numéro-pmr étant le numéro affecté au PMR par le service logiciel.

7. Transférez le fichier sur une disquette (ou un autre volume portable) via, par exemple :

   ```
   # tar -cvf /dev/fd0 numéro-pmr.tarbin.Z
   ```

8. Étiquetez le volume portable en indiquant :

   – le numéro PMR

   – la date de collecte des informations

   – la commande et les indicateurs requis pour extraire les données du volume portable, par exemple :

   ```
   # tar -xvf /dev/fd0
   ```

   Envoyez les données au service de maintenance logicielle.

Annexe A. Commandes de contrôle et d'optimisation des performances AIX

Les outils relatifs aux performances dans l'environnement AIX relèvent de l'une des deux catégories suivantes : informations sur le cours des événements ou moyens d'action sur ces événements. Peu participent des deux catégories. Cette annexe récapitule les commandes relatives aux performances. Nombre d'entre elles sont décrites dans les chapitres traitant de l'optimisation d'aspects spécifiques du système. Pour la majorité, vous trouverez le détail de leur fonction et leur syntaxe dans le manuel *AIX Commands Reference*. Les commandes **schedtune**, **pdt_config**, **pdt_report** et **vmtune** sont décrites plus avant dans cette annexe.

Certaines des commandes relatives aux performances sont fournies comme partie de la boîte à outils Performance Toolbox for AIX (PTX), et non du système d'exploitation de base AIX. Ces commandes sont identifiées par (PTX). Pour déterminer si les outils PTX sont installés, entrez :

```
$ lslpp -lI perfagent.tools
```

Si le module est affiché `AVAILABLE`, les outils PTX sont exploitables.

Les listes suivantes récapitulent les commandes relatives aux performances :

- "Commandes de compte rendu et d'analyse des performances"
- "Commandes d'optimisation", page A-3

Commandes de compte rendu et d'analyse des performances

Ces outils donnent des indications sur les performances d'un ou plusieurs éléments du système, ou sur un ou plusieurs des paramètres ayant une incidence sur les performances.

Commande	Fonction
bf, **bfrpt**	Fournit des rapports détaillés sur les trames d'accès mémoire des applications.
emstat	Rend compte des comptes d'instructions d'émulation.
filemon	Consigne, via l'utilitaire de suivi, l'activité E/S des volumes physiques, des volumes logiques, des fichiers individuels et du gestionnaire VMM (Virtual Memory Manager).
fileplace	Affiche l'emplacement physique ou logique des blocs constitutifs d'un fichier, à l'intérieur du volume physique ou logique dans lequel ils résident.
gprof	Rend compte du flux de contrôle au travers des sous-routines d'un programme et du temps CPU consommé par chaque sous-routine.
iostat	Affiche l'utilisation des données par : • Terminaux • CPU • Disques
lockstat	Affiche des informations relatives aux confits de verrouillage du noyau.

lsattr Affiche les attributs ayant une incidence sur les performances, tels que :

- La taille des caches
- La taille de la mémoire réelle
- Le nombre maximal de pages dans le cache tampon d'E/S bloc
- Le nombre maximal de kilo-octets de mémoire autorisés pour les mbuf
- Les niveaux haut et bas de la régulation des E/S disque.

lslv Affiche des informations sur un volume logique.

netpmon Rend compte de l'activité réseau via l'utilitaire de suivi :

- Consommation CPU
- Débits de données
- Délais de réponse.

netstat Affiche des informations et des statistiques diverses sur les communications, telles que :

- L'état actuel du pool mbuf
- Les tables de routage
- Des statistiques cumulées sur l'activité du réseau.

nfso Affiche (ou modifie) les valeurs des option NFS.

nfsstat Affiche des statistiques sur l'activité client et serveur NFS (Network File System) et RPC (Remote Procedure Call).

no Affiche (ou modifie) les valeurs des options réseau, telles que :

- Tailles par défaut des tampons socket d'envoi et de réception
- Quantité maximale de mémoire utilisée dans le pool mbuf et les pools de grappe.

pdt_config Lance, arrête ou modifie les paramètres de PDT (Performance Diagnostic Tool).

pdt_report Génère un rapport PDT sur la base des données historiques actuelles.

ps Affiche des statistiques et des informations d'état sur les processus du système :

- ID processus
- Activité E/S
- Utilisation de CPU

sar Affiche des statistiques sur les activités du système d'exploitation :

- Accès aux répertoires
- Appels système de lecture et d'écriture
- Duplications et exécutions
- Pagination.

schedtune Affiche (ou modifie) la valeur des paramètres de contrôle de charge de la mémoire VMM, la durée des tranches horaires CPU et l'intervalle de réessai d'une opération n'ayant pas abouti suite à une insuffisance d'espace de pagination.

smit	Affiche (ou modifie) les paramètres de gestion du système.
stem	Prend en charge l'instrumentation des entrées et sorties des programmes exécutables, sans accès au code source de l'exécutable.
svmon	Indique l'état de la mémoire aux niveaux système, processus et segment.
syscalls	Enregistre et compte les appels système.
time	Imprime le délai écoulé et le temps CPU utilisé par l'exécution d'une commande.
tprof	Consigne, via l'utilitaire de suivi, la consommation de CPU par les services de noyau, les sous-routines de bibliothèque, les modules de programmes d'application et des lignes isolées de code source du programme d'application.
trace	Ecrit un fichier qui retrace la séquence exacte des activités du système.
vmstat	Affiche des données VMM : • Nombre de processus en attente ou susceptibles d'être diffusés • Taille de la liste de trames de page disponibles • Activité de défaut de page • Utilisation de CPU.
vmtune	Affiche (ou modifie) les paramètres de l'algorithme de remplacement de page VMM (Virtual Memory Manager).

Commandes d'optimisation

Les outils suivants permettent de modifier un ou plusieurs éléments liés aux performances du système.

Commande	Fonction
fdpr	Optimise des fichiers exécutables pour une charge de travail donnée.
nfso	Affiche (ou modifie) les valeurs des option NFS.
nice	Exécute une commande selon une priorité spécifiée.
no	Affiche (ou modifie) les valeurs des options réseau.
renice	Modifie la priorité des processus en cours.
reorgvg	Réorganise les éléments d'un groupe de volume.
rmss	Réduit temporairement la taille de RAM effective d'un système pour évaluer les performances probables d'un travail sur une machine plus petite ou pour vérifier la taille de mémoire requise par un élément d'un travail.
schedtune	Affiche (ou modifie) la valeur des paramètres de contrôle de charge de la mémoire VMM, la durée des tranches horaires CPU et l'intervalle de réessai d'une opération n'ayant pas abouti suite à une insuffisance d'espace de pagination.

smit	Affiche (ou modifie) les paramètres de gestion du système.
vmtune	Affiche (ou modifie) les paramètres de l'algorithme de remplacement de page VMM (Virtual Memory Manager).

Voir aussi

"Commandes : généralités" dans *AIX 4.3 Guide de l'utilisateur : système d'exploitation et unités* explique comment interpréter les diagrammes de syntaxe.

Commande emstat

Objet

Rend compte des comptes des instructions émulées.

Syntaxe

emstat [*Interval*] [*Count*]

Description

La commande **emstat** fournit des statistiques sur le nombre d'instructions que le système doit émuler. Ce décompte est utilisé pour déterminer si une application doit être recompilée pour éliminer les instructions qui doivent être émulées sur les processeurs PowerPC 601 ou PowerPC 604. Une instruction qui doit être émulée requiert davantage de cycles CPU qu'une instruction qui ne doit pas l'être.

Si une application a été compilée sous AIX version 3, elle peut contenir des instructions non disponibles sur les processeurs 601 ou 604. Le 601 est dotés de la majorité des instructions POWER ; il lui en manque environ cinq, rarement exécutées : les problèmes sont donc exceptionnels. Par contre, il manque environ 35 instructions POWER au processeur 604 : les chances d'émulation sont plus élevées.

Ce phénomène d'instruction émulée se produit avec des applications qui n'ont été compilées en mode commun sous AIX version 3 et sont exécutées sur un processeur 601 ou 604 sans recompilation préalable. Si l'application a été compilée sous AIX version 4.1, l'option de compilation par défaut est "common", aussi seules des instructions communes à tous les processeurs sont intégrés à l'exécutable. Les instructions émulées peuvent également apparaître sur un processeur 601 ou 604 si une application a été compilée pour une architecture spécifique, POWER ou POWER2, par exemple.

La solution est de recompiler l'application, sous AIX version 4.1 ou sous AIX version 3.2.5, en spécifiant l'option qarch=com. Pour effectuer une compilation en mode commun sous AIX version 3.2.5, il vous faut un APAR qui introduise ce code commun.

Le paramètre *Interval* spécifie le délai (en secondes) entre chaque compte rendu. Le premier compte rendu contient des statistiques concernant la durée écoulée depuis le lancement du système. Les comptes rendus suivants contiennent des statistiques collectées au cours de l'intervalle écoulé depuis le compte rendu précédent. Si vous ne spécifiez pas le paramètre *Interval*, la commande **emstat** génère un seul rapport, puis quitte.

Le paramètre *Count* ne peut être spécifié qu'associé au paramètre *Interval*. S'il est spécifié, le paramètre *Count* détermine le nombre de rapports générés. Si le paramètre *Interval* est spécifié sans le paramètre *Count*, les rapports sont générés en continu. Le paramètre *Count* ne peut être affecté de la valeur 0.

Sortie

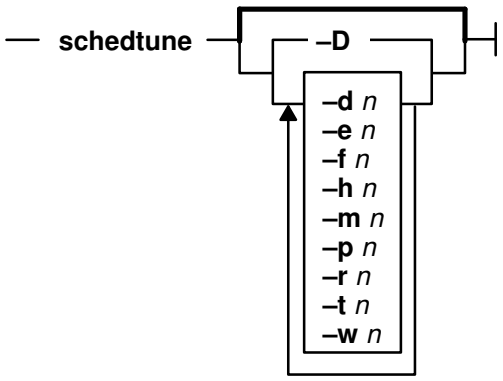
La première colonne est le nombre total d'instructions émulées depuis le dernier réamorçage du système. La deuxième colonne est le nombre d'instructions émulées pendant l'intervalle spécifié.

Commande schedtune

Objet

Définit les paramètres de traitement du programmateur de CPU et du gestionnaire VMM (Virtual Memory Manager).

Syntaxe



schedtune [**–D** | { [**–d** *n*] [**–e** *n*] [**–f** *n*] [**–h** *n*] [**–m** *n*] [**–p** *n*] [**–r** *n*] [**–t** *n*] [**–w** *n*] }]

Description

Paramètres de calcul de priorité

La priorité de la plupart des processus utilisateur varie en fonction du temps CPU récemment utilisé par le processus. Les calculs de priorité du programmateur de CPU sont basés sur deux paramètres définis via **schedtune** : **–r** et **–d**. Les valeurs de *r* et de *d* sont exprimées en unités de trente secondes (1/32) ; c'est-à-dire que la formule utilisée par le programmateur pour calculer la quantité à ajouter à la priorité d'un processus pour pénaliser un usage récent de CPU est la suivante :

pénalité CPU = (valeur de CPU récemment utilisée par le processus) * (*r*/32)

et le recalcul une fois par seconde de la valeur de CPU récemment utilisée par chaque processus est effectué selon la formule :

nouvelle valeur de CPU récemment utilisée = (ancienne valeur) * (*d*/32)

Par défaut, *r* et *d* ont tous deux la valeur 16 – qui assure la comptabilité de la planification de CPU avec les versions antérieures d'AIX. Avant de tester ces valeurs, consultez la section "Optimisation du calcul de la priorité d'un processus via schedtune", page 6-23.

Paramètres de contrôle de charge mémoire

Le programmateur AIX exécute un contrôle de charge mémoire en suspendant les processus lorsque la mémoire est "sur-sollicitée". Le système n'éjecte pas les processus, mais vole les pages au fur et à mesure qu'elles sont requises par la mémoire courante. Normalement, les pages sont volées aux processus suspendus. La mémoire est considérée sur-sollicitée si :

$p * h > s$	*p* étant le nombre de pages écrites dans l'espace de pagination au cours de la dernière seconde, *h* un entier défini par l'indicateur **–h**, et *s* le nombre de vols de pages commis pendant la dernière seconde.

Un processus est suspendu lorsque la mémoire est sur-sollicitée et que :

r * *p* > *f* — *r* étant le nombre de ”repages” accumulées par le processus au cours de la dernière seconde, *p* un entier défini par l'indicateur **–p**, et *f* le nombre de défauts de page rencontrés par le processus au cours de la dernière seconde.

Les processus à priorité fixe de même que les processus noyau ne peuvent être suspendus.

Le terme ”repages” fait référence au nombre de pages appartenant au processus qui ont été réclamées et qui, très vite, ont à nouveau été référencées par le processus.

L'utilisateur peut également spécifier un niveau minimal de multiprogrammation via l'indicateur **–m**. Ceci garantit l'activité d'un minimum de processus pendant la période de suspension. Les processus sont dits actifs lorsqu'ils sont exécutables et qu'ils sont en attente d'une E/S de page. Les processus en attente d'événements et les processus suspendus ne sont pas considérés actifs, pas plus que le processus d'attente.

Les processus suspendus peuvent être réintégrés lorsque le système est resté en deçà du seuil de surcharge pendant *n* secondes, *n* étant spécifié par l'indicateur **–w**. Les processus sont réintégrés au système sur la base de leur priorité, puis de la durée de leur suspension.

Avant de tester ces valeurs, consultez la section ”Optimisation du contrôle de charge de la mémoire VMM”, page 7-14.

Paramètre d'incrément de tranche horaire

La commande **schedtune** permet également de modifier la durée (*tranche horaire*) pendant laquelle le système d'exploitation autorise l'exécution d'un processus, avant d'appeler le répartiteur pour choisir un autre processus à exécuter. Par défaut, cette durée est celle d'une seule impulsion d'horloge (10 millisecondes). Le cas échéant, indiquez le nombre d'impulsions supplémentaires souhaitées via l'indicateur **–t** de la commande **schedtune**.

Sous AIX version 4.1, ce paramètre ne s'applique qu'aux routines assorties de la politique de planification SCHED_RR. Reportez-vous à ”Planification des routines de portée concurrentielle globale ou locale”, page 2-2.

Paramètre d'intervalle de réessai fork()

Si un appel de sous-routine **fork()** échoue suite à un espace de pagination insuffisant pour créer un nouveau processus, le système relance l'appel au bout d'un certain délai. Ce délai est défini par l'indicateur **schedtune –f**.

Restrictions sur schedtune

Seul l'utilisateur `root` est habilité à exécuter **schedtune**. Les modifications effectuées par le biais de la commande **schedtune** ne restent en vigueur que jusqu'à la réinitialisation suivante du système. Pour modifier des paramètres de tranche horaire ou VMM de façon permanente, placez la commande **schedtune** appropriée dans /**etc/inittab**.

> **Attention :** Mal utilisée, cette commande peut induire une détérioration des performances ou une panne du système d'exploitation. Assimilez bien les sections concernées avant de modifier des paramètres système via **schedtune**.

Indicateurs

A défaut d'indicateurs, les valeurs courantes sont imprimées.

–D	Restaure les valeurs par défaut (**d**=16, **e**=2, **f**=10, **h**=6, **m**=2, **p**=4, **r**=16, **t**=0, **w**=1).
–d *n*	La quantité de CPU récemment utilisée par chaque processus est multipliée par *d*/32 une fois par seconde.
–e *n*	Indique qu'un processus suspendu récemment puis réintégré peut être à nouveau suspendu s'il a été actif pendant au moins *n* secondes.
–f *n*	Nombre d'impulsions d'horloge (de 10 millisecondes) définissant le délai avant relance d'un appel **fork** inabouti par insuffisance d'espace de pagination. Le système effectue jusqu'à cinq tentatives de relance de **fork**.
–h *n*	Spécifie le critère système global de déclenchement et d'arrêt de la suspension des processus. La valeur zéro désactive effectivement le contrôle de charge de la mémoire.
–m *n*	Définit le niveau minimal de multiprogrammation.
–p *n*	Spécifie le critère par processus de détermination du processus à suspendre.
–r *n*	La quantité de CPU récemment utilisée par un processus est multipliée par *r*/32 lorsque la priorité du processus est recalculée.
–t *n*	Accroît la durée d'une tranche horaire – délai maximal au bout duquel est planifiée l'exécution d'un autre processus. La durée par défaut est de 10 millisecondes. Le paramètre *n* est exprimé en unités de 10 millisecondes. Si *n* = 0, la durée de la tranche horaire est de 10 millisecondes. Si n = 2, elle est de 30 millisecondes. Sous AIX version 4.1, ce paramètre ne s'applique qu'aux routines assorties de la politique de planification SCHED_RR.
–w *n*	Délai (en secondes) à respecter après la fin d'un emballement, avant réactivation des processus suspendus.
–?	Affiche une brève description de la commande et de ses paramètres.

Voir aussi

La sous-routine **fork**.

”Gestion de la mémoire réelle”.

”Optimisation du calcul de la priorité des processus avec **schedtune**”, page 6-23.

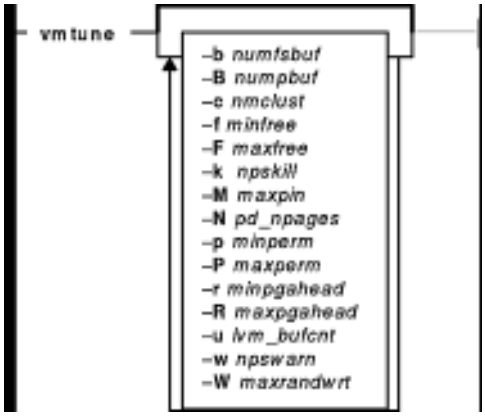
”Optimisation du contrôle de charge de la mémoire VMM”, page 7-14.

Commande vmtune

Objet

Modifie les paramètres d'exploitation du gestionnaire de mémoire virtuel (VMM) et d'autres composants d'AIX.

Syntaxe



vmtune [**–b** *numfsbuf*] [**–B** *numpbuf*] [**–c** *numclust*] [**–f** *minfree*] [**–F** *maxfree*] [**–k** *npskill*] [**–l** *lrubucket*] [**–M** *maxpin*] [**–N** *pd_npages*] [**–p** *minperm*] [**–P** *maxperm*] [**–r** *minpgahead*] [**–R** *maxpgahead*] [**–u** *lvm_budcnt*] [**–w** *npswarn*] [**–W** *maxrandwrt*]

Description

Le gestionnaire de mémoire virtuelle (VMM) tient à jour la liste de trames de page libres de la mémoire réelle. Ces trames sont disponibles pour les pages de mémoire virtuelle, requises pour pallier un défaut de page. Lorsque le nombre de pages de la liste des disponibilités passe en dessous du seuil spécifié par le paramètre *minfree*, VMM commence à voler des pages pour les ajouter à la liste des disponibilités. VMM continue à voler des pages jusqu'à ce la liste des disponibilités ait au moins le nombre de pages spécifié par le paramètre *maxfree*.

Si le nombre de pages de fichier (pages permanentes) en mémoire est inférieur au nombre spécifié par le paramètre *minperm*, VMM vole des trames aux pages de fichier ou calculées, quels que soient les taux de pages déjà référencées (”repages”). Si ce nombre est supérieur à *maxperm*, VMM ne vole que des pages de fichier. Entre les deux, VMM ne vole normalement que des trames de pages de fichier, mais si le taux de pages de fichier déjà référencées est supérieur à celui des pages calculées, les pages calculées sont également volées.

Si un processus lit un fichier séquentiellement, la valeur de *minpgahead* détermine le nombre de pages à lire par anticipation dès détection de la condition. La valeur de *maxpgahead* définit le nombre maximal de pages lues par anticipation, quel que soit le nombre de lectures séquentielles antérieures.

Sous la version 3.2.5, il est impossible de fixer plus de 80 % de la mémoire réelle. Sous AIX version 4.1, le paramètre *maxpin* permet de fixer le pourcentage limite de mémoire fixe.

AIX version 4.1 permet d'optimiser le nombre de systèmes de fichiers `bufstructs` (*numfsbuf)* et la quantité de données traitées par l'algorithme d'écriture différée (*numclust*).

Sous AIX version 4.1, vous pouvez également modifier les seuils déterminant l'insuffisance d'espace de pagination. Le paramètre *npswarn* spécifie le nombre de pages de l'espace de pagination disponibles, à partir duquel le système commence à avertir les processus que l'espace de pagination est presque saturé. Le paramètre *npskill* spécifie le nombre de

pages de l'espace de pagination à partir duquel le système commence à tuer les processus pour libérer de l'espace.

Seul l'utilisateur `root` est habilité à exécuter **vmune**. Les modifications effectuées par le biais de la commande **vmune** ne restent en vigueur que jusqu'à la réinitialisation suivante du système. Pour modifier des paramètres VMM de façon permanente, placez la commande **vmune** appropriée dans **inittab**.

Attention : Mal utilisée, cette commande peut induire une détérioration des performances ou une panne du système d'exploitation. Avant de modifier des paramètres système via **vmtune**, reportez-vous à "Performances du gestionnaire de mémoire virtuelle (VMM)", page 2-5 et à "Optimisation du remplacement de page VMM", page 7-16.

Indicateurs

–b *numfsbuf*	Nombre de systèmes de fichiers `bufstruct`. Valeur par défaut :
–B *numpbuf*	Nombre de pbufs utilisés par LVM. Valeur maximale : Sous AIX version 3, le nombre de pbufs doit parfois être augmenté sur les systèmes effectuant souvent de vastes opérations d'E/S séquentielles.
–c *numclust*	Nombre de grappes de 16 ko traitées en écriture différée. Valeur par défaut :
–f *minfree*	Nombre minimal de trames de la liste de disponibilités. Valeur comprise entre 8 et 204 800.
–F *maxfree*	Nombre de trames sur la liste de disponibilités à partir duquel le vol de pages est interrompu. Sa valeur est comprise entre 16 et 204 800, mais doit être supérieure au nombre spécifié par *minfree* d'une valeur au moins égale à *maxpgahead*.
–k *npskill*	Nombre de pages disponibles dans l'espace de pagination, à partir duquel AIX commence à tuer les processus. Valeur par défaut :
–l *lrubucket*	Taille (en pages de 4 ko) du compartiment de repositionnement de page le moins récemment utilisé (lru). Il s'agit du nombre de trames de page qui pourront être examinées en une seule opération en vue de déchargements possibles lorsqu'une trame libre est requise. Un nombre peu élevé se traduit par un temps de latence faible, mais engendre cependant un comportement quelque peut différent de celui d'un véritable algorithme lru. La valeur par défaut est de 512 Mo et le minimum est de 256 Mo. L'optimisation de cette option n'est pas conseillée.
–M *maxpin*	Pourcentage maximal de mémoire réelle susceptible d'être fixée. Valeur par défaut : Si vous modifiez cette valeur, veillez à conserver au moins 4 Mo de mémoire non fixe pour le noyau.

–N *pd_npages*	Nombre de pages à supprimer d'un coup de la RAM lorsqu'un fichier est supprimé. Valeur par défaut : la plus grande taille de fichier possible divisé par la taille de page (actuellement 4096). Si la taille maximale de fichier est 2 Go, la valeur par défaut de *pd_npages* est 524288. Optimiser cette option ne présente d'intérêt que pour les applications en temps réel.
–p *minperm*	Point en-deçà duquel les pages de fichier sont protégées de l'algorithme de page déjà référencées. Cette valeur est un pourcentage du total de trames de pages de mémoire réelle du système. La valeur spécifiée doit être supérieure ou égale à 5.
–P *maxperm*	Point au-delà duquel l'algorithme de vol de pages ne vole que des pages de fichier. Cette valeur est un pourcentage du total de trames de pages de mémoire réelle du système. La valeur spécifiée doit être supérieure ou égale à 5.
–r *minpgahead*	Nombre de pages à partir duquel commence la lecture séquentielle anticipée. Cette valeur doit être une puissance de 2, comprise entre 0 et 4 096.
–R *maxpgahead*	Nombre maximal de pages à lire par anticipation. Cette valeur doit être une puissance de 2, comprise entre 0 et 4 096, supérieure ou égale à *minpgahead*.
–u *lvm_bufcnt*	Nombre de tampons LVM pour les E/S physiques brutes. Valeur par défaut : Valeurs possibles comprises entre 1 et 64. Cette option n'est disponible que sous AIX version 4.1.
–w *npswarn*	Nombre de pages libres de l'espace de pagination à partir duquel AIX commence à envoyer le signal SIGDANGER aux processus. Valeur par défaut :
–W *maxrandwrt*	Seuil (en pages de 4 ko) d'écritures aléatoires à accumuler en RAM avant qu'elles soient synchronisées sur disque via un algorithme d'écriture différée. Ce seuil est défini sur la base des fichiers. L'option **–W** *maxrandwrt* n'est disponible que sous AIX version 4.1.3 et ultérieures. La valeur par défaut de *maxrandwrt* est 0 (désactivation de l'écriture différée aléatoire).

Voir aussi

"Performances du gestionnaire de la mémoire virtuelle (VMM)", page 2-5.

Script pdt_config

Objet

Contrôle les opérations de l'outil PDT (Performance Diagnostic Tool).

Syntaxe

pdt_config

Description

Le script **pdt_config** est interactif. Appelé, il affiche le menu :

```
# /usr/sbin/perf/diag_tool/pdt_config
 _________________PDT customization menu____________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number:
```

Pour sélectionner une option, tapez son numéro et appuyez sur Entrée.

Si le répertoire **/usr/sbin/perf/diag_tool** ne se trouve pas dans le chemin de recherche, le script peut être appelé par **/usr/sbin/perf/diag_tool/pdt_config**.

Seul l'utilisateur `root` est habilité à exécuter le script **pdt_config**.

Indicateurs

Aucun.

Voir aussi

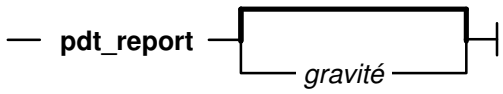
Chapitre 12. "Outil de diagnostic PDT".

Script pdt_report

Objet

Génère un compte rendu PDT (Performance Diagnostic Tool) sur la base des informations historiques actuelles.

Syntaxe



pdt_report [*severity* **]**

Description

PDT contrôle régulièrement les performances du système et ajoute les données à une base de données historique. Normalement, PDT génère un rapport par jour, à une heure définie. Le script **pdt_report** crée ce compte rendu sur demande. Le compte rendu est écrit sur **stdout**. Les messages d'erreur sont dirigés vers **stderr**.

Les messages issus de PDT peuvent être de gravité 1 à 3 (1 étant le niveau le plus élevé). Par défaut, seuls les messages de gravité 1 sont intégrés au compte rendu. Mais vous pouvez commander à **pdt_report** d'include des messages de moindre gravité.

Si le répertoire /**usr**/**sbin**/**perf**/**diag_tool** ne se trouve pas dans le chemin de recherche, le script peut être appelé par /**usr**/**sbin**/**perf**/**diag_tool**/**pdt_report**.

Indicateurs

gravité	Niveau de gravité à partir duquel les messages sont intégrés au rapport. Valeur : de 1 à 3.

Voir aussi

Chapitre 12. "Outil de diagnostic PDT".

Annexe B. Sous-routines liées aux performances

Voici les routines susceptibles d'aider au contrôle et à l'optimisation des performances :

Sous-routines	Fonction
getpri	Détermine la priorité de programmation d'un processus en cours d'exécution.
getpriority	Détermine la valeur nice d'un processus en cours d'exécution.
getrusage	Extrait des informations sur l'utilisation de ressources système.
nice	Incrémente la valeur nice du processus courant.
psdanger	Extrait des informations sur l'utilisation de l'espace de pagination.
setpri	Attribue une priorité fixe à un processus en cours d'exécution.
setpriority	Définit la valeur nice d'un processus en cours d'exécution.

Voir aussi

"Commandes de contrôle et d'optimisation des performances AIX"

Annexe C. Mémoire cache et adressage

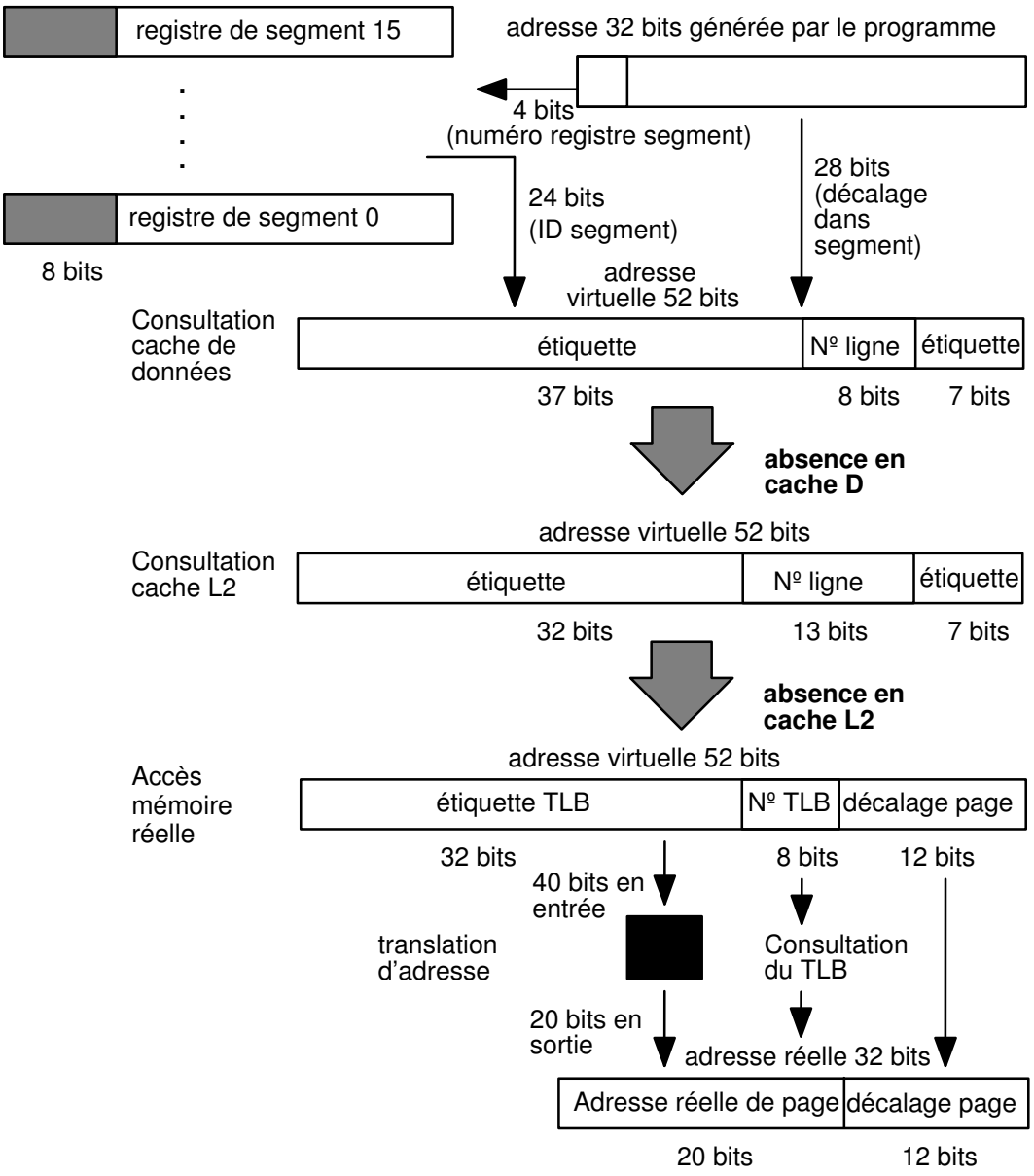
Une exploitation efficace des caches constituant un élément essentiel des performances d'un processeur, les développeurs de logiciels ont intérêt à bien maîtriser les techniques de codage, envisagées sous l'angle de l'utilisation du cache. Bien comprendre cet aspect du problème suppose quelques connaissances sur les architectures de cache ESCALA.

Préliminaire

Les sections qui suivent s'adressent aux programmeurs soucieux de connaître l'impact de la mise en mémoire cache et de l'adressage virtuel sur les performances de leurs programmes. Les ingénieurs intéressés par la logique électronique et la conception précise du ESCALA n'y trouveront sans doute pas leur compte : cette présentation ne prétend pas à l'exhaustivité et la distinction entre les architectures POWER, PowerPC et POWER2 en est quasiment absente.

Adressage

La figure "Transformations successives d'une adresse mémoire" (section "Consultation du cache") illustre les étapes par lesquelles une adresse 32 bits en mémoire virtuelle, générée par un programme, se transforme en adresse mémoire réelle. Le nombre exact de bits varie selon le modèle. Les modèles diffèrent dans leurs détails, mais pas dans le principe de leur conception.

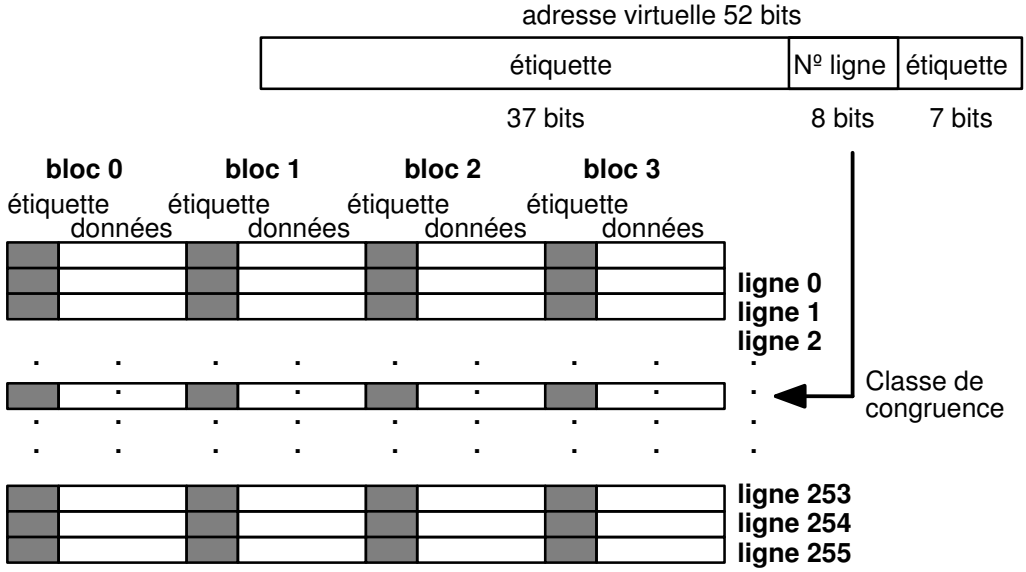


Transformations successives d'une adresse mémoire

Lorsqu'un programme demande le chargement d'un registre avec le contenu d'une portion de la mémoire, l'emplacement mémoire est spécifié par le biais d'une adresse virtuelle de 32 bits. Les 4 bits supérieurs de cette adresse servent à indexer le bloc de registres de 16 segments. Les registres de segment, maintenus par le système d'exploitation, contiennent à tout moment l'ID segment de 24 bits affecté au processus en cours d'exécution. Ces ID segment sont uniques, à moins qu'un processus ne partage un segment avec un ou plusieurs autres processus. L'ID segment 24 bits du registre de segment sélectionné est combiné avec les 28 bits inférieurs de l'adresse des données pour former l'adresse virtuelle 52 bits de l'élément de données à charger. Le décalage dans le segment étant de 28 bits, chaque segment a une longueur de 256 Mo.

Consultation du cache

L'adresse virtuelle 52 bits est utilisée pour la consultation du cache de données, comme illustré à la figure "Consultation d'un cache de données". Les lignes du cache ayant une longueur de 128 octets, les 7 bits inférieurs de l'adresse représentent le décalage à l'intérieur de la ligne du cache. Le cache de données, qui contient 128 ko d'espace, est associatif à quatre voies. Chaque bloc du cache contient ainsi 256 lignes de 128 octets (128 ko/(128*4) = 256), et les 8 bits supérieurs suivants représentent le numéro de ligne (0-255). Chaque bloc du cache possède une ligne portant ce numéro, et les quatre lignes dotées du même numéro forment une classe de congruence, c'est-à-dire les quatre emplacements possibles des données cherchées. Il s'agit d'un *cache associatif à quatre voies.* Si la classe de congruence comportait deux membres, nous aurions parlé d'un cache associatif à deux voies. S'il y avait exactement une ligne de cache par adresse, le cache serait dit en *mappage direct*.



consultation cache données

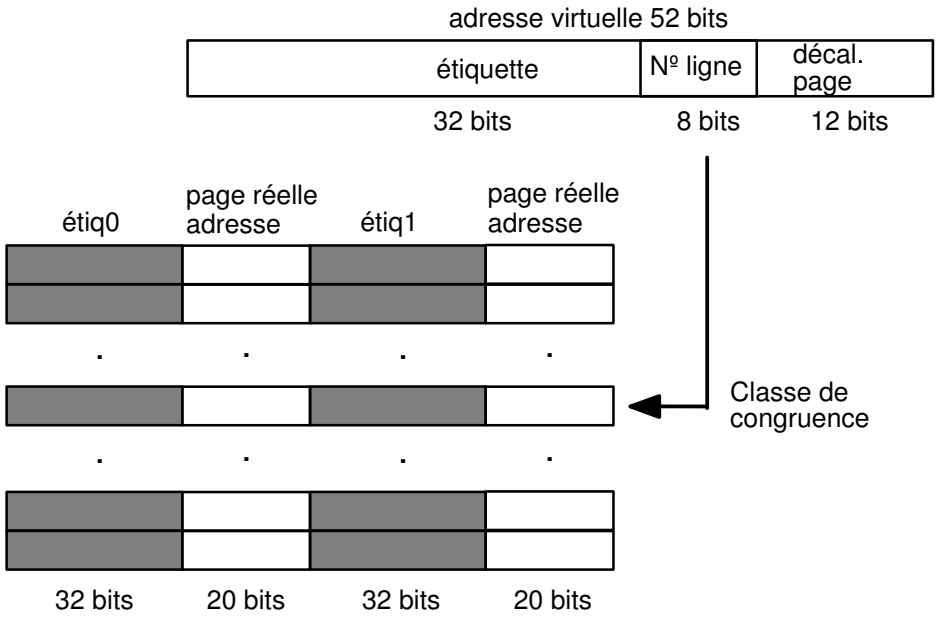
A chaque ligne du cache est associée une étiquette de 37 bits, constituée des bits supérieurs de l'adresse 52 bits à partir de laquelle la ligne cache a été chargée à l'origine. Si une des étiquettes des quatre lignes de l'ensemble de congruence correspond aux 37 bits supérieurs de l'adresse virtuelle 52 bits qui vient d'être générée, il y a *présence en cache.* Les données de la ligne sont chargées dans le registre, et aucun accès RAM (et donc aucune adresse réelle) n'est requis.

Si aucune des étiquettes de l'ensemble de congruence ne correspond à l'étiquette des données à charger, il y a absence de données en cache. Cette machine comporte un cache L2 : la consultation d'un cache est semblable à celle effectuée dans un cache de données. La principale différence entre les deux consultations de caches est que le cache L2 est en mappage direct. Les lignes ont une longueur de 128 octets, et le cache offre une capacité de 1 Mo : il y a donc 8192 lignes. Les 7 bits inférieurs de l'adresse 52 bits constituent toujours le décalage dans la ligne. Les 13 bits suivants constituent le numéro de la ligne du cache. A chaque ligne est associée une étiquette 32 bits unique. Si cette étiquette correspond aux 32 bits supérieurs de l'adresse 52 bits, il y a présence en cache L2. Sinon, l'adresse réelle des données doit être déterminée et les données extraites de la RAM.

Différentes implantations des architectures POWER présentent des caches de taille et de géométrie variables ; certaines sont dépourvues de cache L2, d'autres sont dotées de caches combinant instructions et données, d'autres encore ont des longueurs de lignes différentes. Taille et position des champs de l'adresse 52 bits peuvent différer quelque peu, mais les principes de consultation du cache restent les mêmes.

Consultation du TLB

Le tampon de données TLB (translation lookaside buffer) est un cache d'adresses. L'étiquette TLB est constituée des 32 bits supérieurs de l'adresse virtuelle 52 bits. Les 8 bits suivants de cette adresse indiquent le numéro de ligne dans le TLB, qui a 512 entrées et constitue un ensemble associatif à deux voies (chaque bloc est ainsi constitué de 256 entrées). Les 12 bits inférieurs de l'adresse 52 bits définissent le décalage dans la page de 4096 octets. La partie données de chaque ligne du TLB constitue les 20 bits supérieurs de l'adresse de page réelle 32 bits (voir la figure "Consultation du TLB de données"). S'il y a présence en TLB, les 20 bits supérieurs de l'entrée TLB sont combinés avec les 12 bits inférieurs du décalage de page, pour former l'adresse réelle 32 bits des données.



Consultation du TLB de données

En cas d'absence en TLB, le matériel détermine l'adresse réelle des données en faisant appel aux tableaux de pages, via un algorithme (qui n'entre pas dans le cadre de ce manuel). Cette opération consomme plusieurs dizaines de cycles processeur. Une fois calculée l'adresse réelle 32 bits, la portion adresse de page (20 bits) est mise en mémoire cache dans l'entrée TLB ad hoc, et l'étiquette de cette entrée, mise à jour en conséquence.

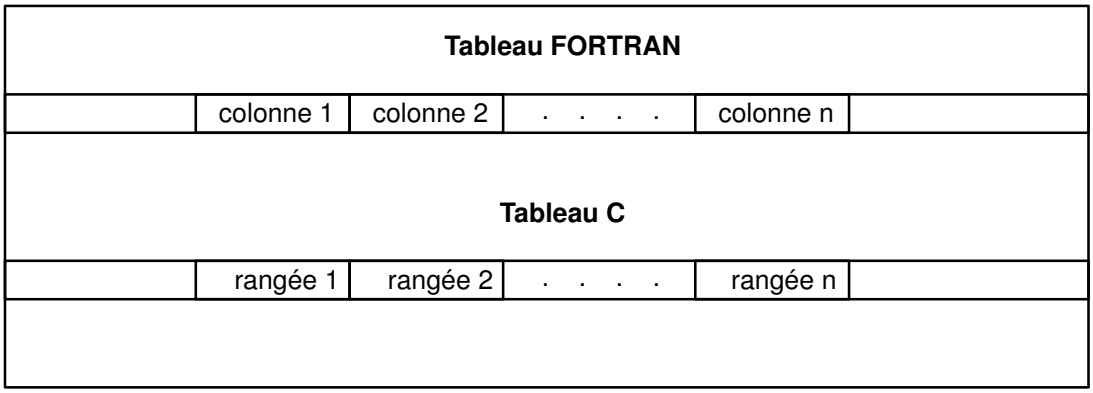
Accès RAM

Quel que soit le moyen utilisé pour l'obtenir, l'adresse 32 bits réelle des données sert à émettre une requête à la RAM. Normalement, un délai de latence d'au moins huit cycles processeur intervient entre l'émission de la requête et le renvoi des 16 premiers octets de données (128 bits, correspondant à la largeur du bus mémoire), incluant les données en cours de chargement. A ce stade, le processeur peut reprendre ses opérations. L'accès RAM se poursuit pendant encore sept cycles processeur, pour charger la ligne complète de cache (128 octets, par tranches de 16 octets). Ainsi, une absence en cache mobilise au moins 16 cycles processeur. L'étiquette de la ligne de cache est mise à jour via les 37 bits supérieurs de l'adresse des données. Le contenu initial de la ligne de cache est perdu.

Implications

Plusieurs types de schémas d'adressage pathologique peuvent provoquer d'incessantes absences en cache ou en TLB, ralentissant notablement la vitesse d'exécution. Si, par exemple, le programme accède à un tableau plus grand que le cache, l'écart étant d'exactement 128 octets, une absence en cache aura lieu à chaque accès. Si le programme présente au processeur une série de requêtes concernant le même numéro de ligne cache, mais dans des pages différentes, une série de collisions entre ensembles de congruence se produiront, entraînant de nombreuses absences en cache même si celui-ci n'est pas saturé. Le fait que le cache soit associatif à quatre voies rend cette éventualité assez improbable, mais il suffit d'un choix malheureux de décalages pour les éléments de données pour ralentir considérablement un programme.

Les grands tableaux peuvent également être à l'origine de problèmes. La figure "Stucture d'un tableau en mémoire" illustre le stockage de tableaux en C et en FORTRAN. Les tableaux C sont basés sur les rangées, tandis que les tableaux FORTRAN le sont sur les colonnes. Si une boucle interne d'un programme C indexe par colonne (ou par rangée pour un programme FORTRAN), un tableau assez grand (512 x 512 en virgule flottante double précision, par exemple) peut provoquer une absence en TLB sur *chaque* accès. Pour en savoir plus sur ces phénomènes, reportez-vous à "Exemple théorique", page 6-10.



Structure d'un tableau en mémoire

Annexe D. Commande ld

L'éditeur de liens AIX (appelé à l'étape finale d'une compilation ou directement via la commande **ld**) offre des fonctions non proposées par l'éditeur de liens UNIX classique. Il peut en résulter un délai d'édition de liens plus long, si la puissance de l'éditeur AIX est inexploitée. Cette annexe présente quelques techniques permettant de l'exploiter au mieux.

Exécutables rééditables

La documentation de l'éditeur de liens fait référence à sa capacité d'accepter un exécutable (un *module chargeable*) comme entrée. Cette fonction peut sensiblement améliorer les performances d'un système où le développement de logiciels constitue une part importante de l'activité, ainsi que les temps de réponse des **ld** individuels.

Sur la majorité des systèmes UNIX classiques, l'entrée de la commande **ld** est toujours constituée d'un ensemble de fichiers contenant du code objet, fichiers individuels **.o** ou fichiers **.o** archivés en bibliothèque. La commande **ld** résout ensuite les références externes de ces fichiers et écrit un exécutable portant par défaut le nom **a.out**. Le fichier **a.out** ne peut être qu'exécuté. Si une erreur est détectée dans l'un des modules intégrés au fichier **a.out**, il faut modifier et recompiler le code source défectueux, puis répéter l'intégralité du processus **ld**, à partir de l'ensemble complet de fichiers **.o**.

Sous AIX, l'éditeur de liens accepte en entrée aussi bien les fichiers **.o** que les fichiers **a.out**, car l'éditeur intègre les dictionnaires de résolution ESD (External Symbol Dictionary) et RLD (Relocation Dictionary), dans le fichier exécutable. Ce qui signifie que l'utilisateur peut rééditer les liens d'un exécutable existant pour remplacer un seul fichier **.o** modifié, au lieu de reconstruire un nouvel exécutable depuis le début. Le processus d'édition consommant des cycles processeur et de l'espace de stockage en partie proportionnellement au nombre de fichiers faisant l'objet d'un accès et au nombre de références aux symboles à résoudre, rééditer les liens d'un exécutable avec une nouvelle version d'un seul module est bien plus rapide que de repartir du début.

Bibliothèques de sous-routines prééditées

Dans certains environnements, la capacité de prééditer les liens de l'intégralité d'une bibliothèque de sous-routines est tout aussi importante. Les bibliothèque de sous{TAG}routines du système telles que **libc.a** sont, en effet, livrées en format d'édition de sortie, et non sous forme d'un fichier archive de fichiers **.o** . L'utilisateur gagne de ce fait un temps considérable de traitement lorsqu'il édite les liens d'une application avec les bibliothèques système requises, dans la mesure où seules les références de l'application aux sous–routines de la bibliothèque doivent être résolues. Les références entre routines de bibliothèques système ont été d'ores et déjà résolues au cours du processus de constitution du système.

Nombre de bibliothèques de sous-routines tierces sont toutefois souvent livrées sous forme d'archive de fichiers **.o** bruts. Lorsqu'un utilisateur édite les liens d'une application avec des bibliothèques de ce type, l'éditeur de liens doivent résoudre les symboles de toute la bibliothèque à chaque édition de liens de l'application. L'opération est d'autant plus longue si les bibliothèques sont volumineuses et les machines peu puissantes.

L'écart de performance entre des bibliothèques prééditées et non prééditées est énorme, notamment sur de petites configurations. Avec préédition de la bibliothèque de sous-routines, l'édition des liens du programme FORTRAN est tombée à environ 1,7 minute. Lorsque le fichier **a.out** résultat a été soumis à une nouvelle édition de liens

avec un nouveau fichier FORTRAN **.o**, pour simuler une correction d'erreur, le délai est passé à environ 4 secondes.

Exemples

1. Pour prééditer les liens d'une bibliothèque, lancez la commande suivante sur le fichier archive :

   ```
   ld -r libfoo.a -o libfooa.o
   ```

2. Compiler et éditer les liens du programme FORTRAN **something.f** revient alors à :

   ```
   xlf something.f libfooa.o
   ```

   Notez que la bibliothèque prééditée est traitée comme un fichier d'entrée normal, et *non* avec la syntaxe habituelle d'identification des bibliothèques (**–lfoo**).

3. Pour recompiler le module et rééditer les liens de l'exécutable après correction d'une erreur, lancez :

   ```
   xlf something.f a.out
   ```

4. Si, toutefois, la correction de l'erreur a conduit à l'appel d'une autre sous-routine de la bibliothèque, l'édition de liens n'aboutira pas. Le script shell Korn suivant teste le retour d'un code d'erreur et y remédie :

   ```
   # !/usr/bin/ksh
   # Shell script for source file replacement bind
   #
   xlf something.f a.out
   rc=$?
   if [ "$rc" != 0 ]
   then
      echo "New function added ... using libfooa.o"
      xlf something.o libfooa.o
   fi
   ```

Voir aussi

Commande **ld**.

Format de fichier Objet XCOFF (a.out).

Annexe E. Outils de performance

De temps en temps, le groupe Performance AIX se voit demander des précisions sur la ”surcharge” occasionnée par les outils de performance. C’est une question intéressante, dans la mesure où certains outils ont une répercussion certaine sur la charge du système. C’est également une question à laquelle il est difficile de répondre, car le coût induit par la plupart des outils est souvent proportionnel à un aspect particulier de la charge du système. Les sections suivantes proposent des considérations succintes et informelles sur la vitesse et les ressources utilisées par les principaux outils de contrôle et d’optimisation. Ces considérations visent à donner une idée du coût relatif des différents outils : elles ne constituent pas une description rigoureuse des performances des outils.

filemon L’essentiel de la charge de **filemon** sur le système est contituée de temps CPU. Dans un environnement saturé au niveau de la CPU avec peu d’E/S, **filemon** a ralenti une longue compilation d’environ 1 %. Dans un environnement saturé au niveau du CPU avec un fort taux de sorties disque, **filemon** a ralenti le programme d’écriture d’environ 5 %.

fileplace La plupart des variantes de cette commande utilisent moins de 0,3 seconde de temps CPU.

iostat Cette commande utilise environ 20 millisecondes de temps CPU pour chaque compte rendu périodique généré.

lsattr Cette commande est limitée au niveau des E/S. A sa première exécution, elle peut demander entre 2 et 4 secondes pour lire les données requises. Les exécutions suivantes, sur un système peu chargé, consomment environ 0,5 seconde de temps CPU.

lslv Cette commande est limitée au niveau de la CPU. Par exemple, la commande :

```
lslv -p hdisk0 hd1
```

consomme environ 0,5 seconde de temps CPU.

netpmon Avec une charge modérée, orientée réseau, **netpmon** augmente la consommation de CPU de 3 à 5 %. Dans un environnement saturé au niveau du CPU avec peu d’E/S, **netpmon** a ralenti une longue compilation d’environ 3,5 %.

netstat La plupart des variantes de cette commande utilisent moins de 0,2 seconde de temps CPU.

nfsstat La plupart des variantes de cette commande utilisent moins de 0,1 seconde de temps CPU.

PDT La collecte quotidienne des données prend plusieurs minutes, mais l’essentiel de ce temps est du temps de ”veille”. La consommation réelle de CPU ne dépasse généralement pas 30 secondes.

ps Le temps CPU consommé par cette commande dépend du nombre de processus à afficher, mais ne dépasse généralement pas 0,3 seconde.

svmon La commande **svmon –G** consomme environ 3,2 secondes de temps CPU. Une commande **svmon** pour un seul processus (**svmon –P** *id-processus*) consomme environ 0,7 seconde de temps CPU.

tprof	**tprof** utilisant **trace**, il peut quelque peu surcharger le système. **tprof** n'active toutefois qu'un point d'ancrage de suivi, la surcharge induite est donc moindre que lors d'un suivi complet. Par exemple, **tprof** a provoqué une baisse de performance d'environ 2 % pour une longue compilation.
trace	La surcharge induite par **trace** dépend largement de la charge du système et du nombre d'ID de points d'ancrage collectés. A l'extrême, un travail long et très mobilisateur de CPU, exécuté sur un système libre par ailleurs, est rallongé d'environ 3,2 % si **trace** est actif et que tous les points d'ancrage le sont aussi.
vmstat	Cette commande utilise environ 40 millisecondes de temps CPU pour chaque compte rendu généré. La commande **vmstat –s** consomme environ 90 millisecondes de temps CPU.

Annexe F. Gestion de la mémoire des applications

AIX a été doté, à partir de la version 3.2, d'un nouvel algorithme de gestion de la mémoire, qui est resté en vigueur avec la version 4. L'algorithme précédent, largement exploité sur les systèmes UNIX, arrondissait la taille de toutes les requêtes **malloc** à la puissance de 2 immédiatement supérieure. Il en résultait une fragmentation considérable de la mémoire – réelle et virtuelle – et un faible regroupement référentiel. Le nouvel algorithme affecte la quantité exacte d'espace requis et est plus efficace pour ce qui concerne la récupération des blocs mémoire précédemment utilisés.

Malheureusement, un certain nombre de programmes d'application existants dépendent du précédent algorithme – au niveau de leurs performances ou même de leur fonctionnement. Par exemple, si un programme dépend de l'espace supplémentaire fourni par l'opération d'arrondi, car il déborde en réalité de la fin d'un tableau, il échouera probablement avec la version 3.2 de **malloc**.

Autre exemple : du fait de l'inefficacité de la récupération d'espace de la routine de la version 3.1, le programme d'application reçoit presque toujours de l'espace qui a été réinitialisé à zéro (lorsqu'un processus touche une page donnée de son segment de travail pour la première fois, cette page est définie à zéro). L'exécution des applications peut dépendre de cet effet secondaire. En fait, la mise à zéro de l'espace affecté n'est pas une fonction propre à **malloc** et entraîne une pénalité non justifiée au niveau des performances, pour les programmes qui n'initialisent que lorsque requis et peut-être pas à zéro.
La version 3.2 de **malloc** étant plus efficace au niveau de la réutilisation de l'espace, les programmes dépendant de la réception de **malloc** d'espace mis à zéro échoueront probablement avec les versions 3.2 et ultérieures du système.

De même, si un programme réaffecte (via **realloc**) continuellement une structure dans une zone de taille légèrement supérieure, dans la version 3.1, **realloc** peut ne pas avoir besoin de déplacer souvent la structure. Souvent, **realloc** peut faire usage de l'espace supplémentaire fourni par l'opération d'arrondi. Sous la version 3.2, **realloc** aura généralement à déplacer la structure vers une zone légèrement plus grande, parce que quelque chose d'autre a été affecté (via **malloc**) juste au-dessus. Ceci explique l'apparente dégradation des performances de **realloc**, alors qu'il s'agit en réalité de l'émergence d'un coût implicite dans la structure du programme d'application.

Nous avons bien entendu prévu le cas où des programmes AIX existants, ou portés à partir d'autres systèmes UNIX, dépendent d'effets secondaires de la version 3.1 de la sous-routine **malloc**. Dans ces cas, l'algorithme de la version 3.1 peut être rappelé via :

```
MALLOCTYPE=3.1; export MALLOCTYPE
```

A partir de là, tous les programmes exécutés par le shell utiliseront la version antérieure de la sous-routine **malloc**. Donner à `MALLOCTYPE` une valeur autre que `3.1` provoque le retour à la version 3.2.

Voir aussi

Sous-routines **malloc** et **realloc**.

Annexe G. Bibliothèques partagées

Partager une bibliothèque offre parfois l'occasion de négocier mémoire et temps de traitement.

Avantages et inconvénients

Partager une bibliothèque signifie disposer d'une seule copie des routines les plus utilisées et maintenir cette copie dans un segment partagé unique. Ce qui peut réduire de façon significative la taille des exécutables, et économiser ainsi de l'espace disque. En outre, dans la mesure où ces routines sont exploitées par plusieurs processus dans un environnement multi-utilisateur, une routine que vous appelez pour la première fois peut déjà se trouver en mémoire réelle : vous gagnez ainsi le temps de chargement correspondant et économisez la trame de page requise. Autre avantage, les routines ne sont pas liées à l'application de façon statique, mais dynamiquement au chargement de l'application. Ce qui permet aux applications d'hériter automatiquement des modifications, sans recompilation ou réédition des liens.

Partager une bibliothèque peut néanmoins présenter quelques inconvénients. Du point de vue des performances, un "code collant" est requis dans l'exécutable pour accéder au segment partagé. Ce code ajoute un nombre de cycles par appel à un segment de bibliothèque partagée. Plus subtile est la réduction du "regroupement référentiel". Si seul un nombre restreint de routines vous intéressent, vous risquez de les trouver dispersées dans l'espace d'adressage virtuel de la bibliothèque : le nombre de pages auxquelles vous devez alors accéder pour appeler les routines est notablement supérieur à ce qui aurait été le cas si les routines avaient été directement liées à l'exécutable. Une des conséquences est que, si vous êtes le seul utilisateur de ces routines, vous vous heurterez à davantage de défauts de page pour les charger toutes en mémoire réelle. En outre, dans la mesure où davantage de pages sont touchées, la probabilité d'une absence d'instruction en TLB (translation lookaside buffer) est plus grande.

Définition d'exécutables partagés ou non

Par défaut, la commande **cc** est associée à l'option de bibliothèque partagée. Pour passer outre cette valeur, spécifiez l'option **–bnso** comme suit :

```
cc xxx.c –o xxx.noshr –O –bnso –bI:/lib/syscalls.exp
```

Choix de l'option appropriée

A l'évidence, le plus simple pour déterminer si une application est sensible au partage des bibliothèques est de recompiler l'exécutable avec l'option de non-partage. Si les performances s'améliorent notablement, il vous appartient de décider si cette amélioration vaut d'abandonner les autres avantages du partage des bibliothèques. Veillez à mesurer les performances dans un environnement réel : un programme édité en mode non partagé peut être exécuté plus rapidement comme instance simple sur une machine faiblement chargée. Ce même programme, exploité simultanément par plusieurs utilisateurs, peut suffisamment augmenter la consommation de mémoire réelle pour ralentir l'ensemble de la charge de travail.

Voir aussi

Shared Libraries and Shared Memory dans *AIX General Programming Concepts : Writing and Debugging Programs*.

Annexe H. Accès à l'horloge du processeur

Les tentatives de mesure, sous AIX, d'intervalles de temps minimes se heurtent souvent à l'activité d'arrière-plan intermittente du sytème d'exploitation et à la mobilisation de temps de traitement par les routines du système. Une solution à ce problème consiste à accéder directement à l'horloge du processeur pour déterminer le début et la fin des mesures interrompues, à effectuer les mesures plusieurs fois et à filtrer le résultat pour éliminer les périodes coupées par des interruptions.

Dans les architectures POWER et POWER2, l'horloge du processeur est implantée comme paire de registres spéciaux. L'architecture PowerPC définit un registre 64 bits appelé base horaire. Ces registres ne sont accessibles qu'aux programmes en assembleur.

Attention : Les durées mesurées par l'horloge du processeur correspondent à des temps *absolus.* Si une interruption se produit dans l'intervalle, la durée calculée inclut celle du traitement de cette interruption et, éventuellement, celle d'autres processus, avant que le contrôle ne revienne au code chronométré. Autrement dit, la durée issue de l'horloge processeur est une durée *brute,* qu'il ne faut pas utiliser sans vérification.

Sous AIX version 4.1, une paire de sous-routines de bibliothèque a été ajoutée au système pour faciliter l'accès à ces registres et le rendre indépendant de l'architecture. Il s'agit des routines **read_real_time** et **time_base_to_time**. La sous-routine **read_real_time** extrait l'heure courante de la source ad hoc et la stocke sous forme de deux valeurs de 32 bits. La sous-routine **time_base_to_time** vérifie que l'heure est exprimée en secondes et nanosecondes, exécutant au besoin les conversions nécessaires à partir du format base horaire. La séparation des deux fonctions (obtention de l'heure et conversion) a pour but de minimiser la charge d'obtention de l'heure.

L'exemple suivant illustre l'utilisation de ces routines pour mesurer le délai d'exécution d'une portion spécifique de code :

```
#include <stdio.h>
#include <sys/time.h>

int main(void) {
   timebasestruct_t start, finish;
   int val = 3;
   int w1, w2;
   double time;

   /* get the time before the operation begins */
   read_real_time(&start, TIMEBASE_SZ);

   /* begin code to be timed */
   printf("This is a sample line %d \n", val);
   /* end code to be timed   */

   /* get the time after the operation is complete
   read_real_time(&finish, TIMEBASE_SZ);

   /* call the conversion routines unconditionally, to ensure    */
   /* that both values are in seconds and nanoseconds regardless */
   /* of the hardware platform.                                  */
   time_base_to_time(&start, TIMEBASE_SZ);
   time_base_to_time(&finish, TIMEBASE_SZ);

   /* subtract the starting time from the ending time */
   w1 = finish.tb_high - start.tb_high; /* probably zero */
   w2 = finish.tb_low - start.tb_low;
```

```
    /* if there was a carry from low-order to high-order during  */
    /* the measurement, we may have to undo it.                  */
    if (w2 < 0) {
       w1--;
       w2 += 1000000000;
    }

    /* convert the net elapsed time to floating point microseconds */
    time = ((double) w2)/1000.0;
    if (w1 > 0)
       time += ((double) w1)*1000000.0;

    printf("Time was %9.3f microseconds \n", time);
    exit(0);
}
```

Pour réduire la charge d'appel et de retour des routines d'horloge, l'analyste peut essayer de lier les références de jeux d'essais non partagées (voir Annexe G).

S'il s'agit d'évaluer des performances réelles, vous pouvez exécuter plusieurs fois le code à mesurer. Si vous avez chronométré ensemble un certain nombre de répétitions consécutives, vous pouvez calculer le temps moyen, sachant qu'il risque d'inclure le temps de traitement d'interruptions ou d'autres activités. Si les répétitions ont été chronométrées individuellement, étudiez les délais pour évaluer s'il sont "raisonnables", mais pensez que le temps d'exécution des routines de chronométrage est inclus dans chaque mesure. Vous pouvez utiliser les deux techniques et comparer les résultats. En tout état de cause, le choix d'une méthode dépend beaucoup de l'objectif des mesures.

Accès à l'horloge unique d'une architecture POWER

Attention : Les sections qui suivent ne s'appliquent qu'aux architectures POWER et POWER2 (et au microprocesseur 601). Les exemples de code *fonctionnent* correctement sur un système PowerPC (ils ne se "planteront" pas), mais certaines instructions ne seront que simulées. Dans la mesure où l'accès à l'horloge du processeur a pour but de mesurer des durées très précises avec une faible charge, les résultats ainsi obtenus sont de peu d'utilité.

Les architectures de processeur POWER et POWER2 comportent deux registres spéciaux (un registre supérieur et un registre inférieur) qui contiennent une horloge haute résolution. Le registre supérieur contient le temps en secondes et le registre inférieur, un décompte des fractions de seconde, en nanosecondes. La précision réelle du registre inférieur dépend de la fréquence de sa mise à jour, qui est spécifique du modèle.

Routines assembleur d'accès aux registres d'horloge POWER

Le module en assembleur suivant (**timer.s**) fournit des routines (`rtc_upper` et `rtc_lower`) d'accès aux registres supérieur et inférieur de l'horloge.

```
           .globl  .rtc_upper
.rtc_upper: mfspr   3,4         # copy RTCU to return register
            br

           .globl  .rtc_lower
.rtc_lower: mfspr   3,5         # copy RTCL to return register
            br
```

Sous-routine C indiquant l'heure en secondes

Le module ci-dessous (**second.c**) contient une routine C qui appelle les routines **timer.s** pour accéder au contenu des registres supérieur et inférieur, et qui retourne l'heure en secondes, sous forme d'une valeur réelle double précision.

```
double second()
{
  int ts, tl, tu;

  ts = rtc_upper();     /* seconds                          */
  tl = rtc_lower();     /* nanoseconds                      */
  tu = rtc_upper();     /* Check for a carry from           */
  if (ts != tu)         /* the lower reg to the upper.      */
    tl = rtc_lower();   /* Recover from the race condition. */
  return ( tu + (double)tl/1000000000 );
}
```

La sous-routine **second** peut être appelée par une routine C ou FORTRAN.

Remarque : La précision de **second.c** peut varier en fonction du temps écoulé depuis la dernière réinitialisation du système. Plus long est ce délai, plus élevé est le nombre de bits de précision consommée par la partie "secondes entières" du nombre. La technique indiquée au début de cette annexe évite le problème en effectuant la soustraction requise pour obtenir le délai écoulé avant conversion en virgule flottante.

Accès aux registres d'horloge dans les systèmes PowerPC

L'architecture du processeur PowerPC comporte un registre de base horaire de 64 bits, logiquement divisé en deux champs (supérieur et inférieur) de 32 bits (TBU et TBL). La fréquence d'incrémentation de ce registre dépend de l'implantation logicielle et matérielle, et peut parfois varier. Convertir les valeurs de la base horaire en secondes est une tâche bien plus complexe que dans une architecture POWER. Nous vous conseillons vivement d'utiliser les interfaces **read_real_time** et **time_base_to_time** pour obtenir la valeur d'horloge des systèmes PowerPC.

Exemple d'utilisation de la routine second

Voici un exemple (**main.c**) de programme C faisant appel à la sous–routine **second** :

```
#include <stdio.h>
double second();
main()
{
      double t1,t2;

      t1 = second();
      my_favorite_function();
      t2 = second();

     printf("my_favorite_function time: %7.9f\n",t2 - t1);
      exit();
}
```

Voici un exemple (**main.f**) de programme FORTRAN faisant appel à la sous–routine **second** :

```
      double precision t1
      double precision t2

      t1 = second()
      my_favorite_subroutine()
      t2 = second()
      write(6,11) (t2 - t1)
11    format(f20.12)
      end
```

Pour compiler et exploiter **main.c** ou **main.f**, utilisez :

```
xlc -O3 -c second.c timer.s
xlf -O3 -o mainF main.f second.o timer.o
xlc -O3 -o mainC main.c second.o timer.o
```

Annexe I. Support NLS (National Language Support)

Le support AIX NLS (National Language Support) facilite l'exploitation d'AIX dans différents environnements de langues. L'utilisation de NLS n'étant pas sans conséquence sur les performances du système, nous allons en présenter les principales caractéristiques.

NLS permet d'adapter AIX en fonction de la langue et de l'environnement culturel de l'utilisateur. Un *environnement local,* qui combine une langue et un ensemble de spécificités géographiques ou culturelles, est identifié par un nom composé, tel que en_US (anglais américain). A chaque environnement local, sont associés un ensemble de catalogues de messages, des tables de tri et d'autres informations définissant les spécificités de l'environnement. A l'installation d'AIX, l'administrateur système décide des informations locales à installer. Par la suite, les utilisateurs peuvent les adapter à chaque shell en modifiant les variables **LANG** et **LC_ALL**.

Le seul environnement local ne répondant pas à la structure ci-dessus est l'environnement C (ou POSIX). Il s'agit de l'environnement par défaut (utilisé si aucun autre n'est explicitement sélectionné). C'est également l'environnement dans lequel est lancé chaque processus nouvellement généré. Une exécution dans l'environnement C équivaut presque, sous AIX, à une exécution dans l'environnement originel, unilingue, d'UNIX. Il n'existe pas de catalogues de messages C : les programmes qui tentent d'obtenir un message d'un catalogue reçoivent le message par défaut compilé dans le programme. Certaines commandes, telle **sort**, reviennent à leurs algorithmes d'origine, spécifiques de leur jeu de caractères.

Selon nos mesures, les performances de NLS relèvent de trois niveaux possibles. L'environnement C est généralement le plus rapide pour l'exécution des commandes, suivi des environnements mono-octets (alphabet latin), tels que en_US, les environnements multioctets étant les plus lents.

Remarques sur la programmation

Historiquement, le langage C a souffert d'une certaine confusion entre les mots *octet* et *caractère*. Ainsi un tableau déclaré comme `char foo[10]` est un tableau de 10 octets. Or, toutes les langues ne sont pas composées de caractères transcriptibles par un seul octet. Le japonais et le chinois, par exemple, requièrent au moins deux octets par caractère. C'est pourquoi, sous AIX, la distinction est faite entre un *octet* (8 bits de données) et un *caractère* (quantité d'informations requises pour représenter un caractère).

Deux des caractéristiques d'un environnement local sont le nombre maximal d'octets requis pour représenter un caractère et le nombre maximal de positions affichables en sortie qu'un caractère peut occuper. Ces valeurs peuvent être obtenues via les macros **MB_CUR_MAX** et **MAX_DISP_WIDTH**. Si *ces deux* valeurs sont égales à 1, l'équivalence entre octet et caractère est pertinente dans cet environnement. Si l'une des deux valeurs est supérieure à 1, les programmes effectuant un traitement caractère par caractère, ou gardant trace du nombre de positions d'affichage, devront recourir aux fonctions d'internationalisation.

Dans la mesure où les codages multioctet comportent un nombre variable d'octets par caractère, ils ne peuvent être traités comme des tableaux de caractères. Pour programmer efficacement dans le cas où chaque caractère doit être traité, un type de données à taille d'octet fixe, **wchar_t**, a été défini. Une donnée de type **wchar_t** est suffisamment grande pour contenir le format converti de n'importe quel caractère codé. Les programmeurs peuvent ainsi déclarer des tableaux de **wchar_t** et les traiter de façon grosso modo équivalente à un tableau de type **char**, en utilisant les analogies avec les caractères étendus des fonctions **libc** classiques. Malheureusement, la conversion du format multioctet

(dans lequel le texte est rédigé, enregistré sur disque ou saisi à l'écran) au format **wchar_t**, est relativement coûteuse. Il convient donc de ne recourir au type **wchar_t** que si l'efficacité du programme compense largement le coût de la conversion de et vers le format **wchar_t**.

Quelques règles

Faute de connaître les quelques contraintes qui pèsent sur les jeux de caractères multioctet et le minimum de fonctions d'internationalisation qui éliminent le problème, vous risquez d'écrire un programme d'une remarquable lenteur. Voici quelques conseils :

- Dans tous les jeux de caractères pris en charge par BULL, les codes 0x00 à 0x3F sont uniques et réservés aux caractères ASCII standard. Etre unique signifie ici que ces combinaisons n'apparaissent jamais comme élément du codage d'un caractère multioctet. Le caractère nul faisant partie de ce jeu, les fonctions **strlen**, **strcpy** et **strcat** s'appliquent aux chaînes multi- et mono-octet. Le programmeur ne doit pas oublier que la fonction **strlen** renvoie le nombre *d'octets* de la chaîne et non le nombre de *caractères*.
- De même, la fonction de chaîne standard `strchr(foostr, '/')` fonctionne correctement dans tous les environnements, dans la mesure où le signe / (barre oblique) fait partie d'un intervalle du code unique. En réalité, la plupart des délimiteurs standard appartiennent à l'intervalle 0x00 à 0x3F, et l'essentiel de l'analyse peut être effectué sans recours aux fonctions d'internationalisation ou à la conversion au format **wchar_t**.
- La comparaison entre chaînes conduit à une de deux relations : égalité ou inégalité. Pour un test d'égalité, optez pour la fonction standard **strcmp**. En écrivant :

  ```
  if (strcmp(foostr,"a rose") == 0)
  ```

  nous ne recherchons pas le nom `"rose"`, mais cette combinaison exacte de bits. Si `foostr` contient `"rosé"`, cela ne nous intéresse pas.
- Les relations d'inégalité sont testées lorsqu'il s'agit d'ordonner des chaînes selon l'ordre de tri défini pour l'environnement local. Pour ce faire, nous utilisons :

  ```
  if (strcoll(foostr,barstr) > 0)
  ```

  en étant conscient du coût de l'opération de tri sur chaque caractère.
- Tout programme exécuté (via **exec**) est toujours démarré dans l'environnement local C. S'il doit faire appel à une ou plusieurs fonctions d'internationalisation, dont des accès aux catalogues de message, il doit exécuter :

  ```
  setlocale(LC_ALL, "");
  ```

  pour passer à l'environnement local de son processus parent avant tout appel à une fonction d'internationalisation.

Contrôle de l'environnement local

La séquence de commandes :

```
LANG=C
export LANG
```

définit l'environnement local *par défaut* à C (C est utilisé, sauf spécification autre explicite via une autre variable, telle **LC_COLLATE**).

La séquence :

```
LC_ALL=C
export LC_ALL
```

force *toutes* les variables à C, quelles que soient les définitions antérieures.

Pour un relevé des valeurs actuelles des variables locales, tapez `locale`.

Voir aussi

”Gestion des ressources AIX : généralités”.

Commande **sort**.

Annexe J. Récapitulatif des paramètres AIX optimisables

Cette annexe donne la liste des paramètres qui ont une incidence sur les performances. Cette liste est classée dans l'ordre alphabétique.

Augmentation de la tranche horaire

Objet :	Nombre d'impulsions d'horloge de 10 millisecondes duquel augmenter la tranche horaire par défaut (10 millisecondes).
Valeurs :	Par défaut : Intervalle : 0 à n'importe quel entier positif.
Affichage :	**schedtune** La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **schedtune** à /**etc**/**inittab**.
Modification :	**schedtune –t** *nouvelle-valeur*
Diagnostics :	N/A.
Optimisation :	Ce paramètre ne doit normalement pas être modifié. Si la charge de travail est constituée essentiellement de programmes longs et gourmands en CPU, une augmentation de ce paramètre peut avoir un effet positif.
Voir :	"Modification de la tranche horaire du programmateur", page 6-25.

arpt_killc

Objet :	Délai avant suppression d'une entrée ARP inactive et terminée.
Valeurs :	Par défaut : 20 (minutes). Intervalle : N/A.
Affichage :	**no –a** ou **no –o arpt_killc**
Modification :	**no –o arpt_killc=***nouvellevaleur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics :	N/A.
Optimisation :	Pour réduire l'activité ARP dans un réseau stable, vous pouvez augmenter **arpt_killc**. L'effet est peu significatif.
Voir :	N/A.

Calcul de la priorité d'un processus

Objet :	Spécifie la valeur de l'augmentation de la priorité d'un processus en fonction de son utilisation récente de CPU, et le taux de décroissance du temps d'utilisation de CPU récente. Ces paramètres sont appelés *r* et *d*.
Valeurs :	Par défaut : Intervalle : 0 à 32 (**Remarque :** Lors du calcul, les valeurs de *r* et *d* sont divisées par 32. L'intervalle réel des facteurs va ainsi de 0 à 1 par incréments de 0,03125.)
Affichage :	**schedtune**
Modification :	**schedtune –r** ou **schedtune –d** La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **schedtune** à **/etc/inittab**.
Diagnostics :	**ps al** Si vous constatez que la colonne PRI indique des priorités pour les processus d'avant-plan (ceux qui ont des valeurs NI de 20) supérieures aux valeurs PRI de certains processus d'arrière-plan (valeurs NI > 20), vous devez peut-être diminuer la valeur de *r*.
Optimisation :	Diminuer *r* rend plus compétitifs les processus d'avant-plan. Diminuer *d* évite aux processus d'avant-plan d'être en conflit plus longtemps avec les processus d'arrière-plan. **schedtune –r 2** assure à chaque nouveau processus d'avant-plan au moins 0,5 secondes de temps CPU, avant de le laisser en concurrence avec les autres processus dont NI >= 24.
Voir :	Reportez-vous à "Optimisation du calcul de la priorité des processus avec schedtune", page 6-23.

Compte biod

Objet :	Nombre de processus **biod** disponibles pour gérer les requêtes NFS sur un client.
Valeurs :	Par défaut : Intervalle : 1 à n'importe quel entier positif.
Affichage :	**ps –ef | grep biod**
Modification :	**chnfs –b** *nouvellevaleur* Le changement prend effet immédiatement et est permanent. L'indicateur **–N** entraîne un changement immédiat, mais temporaire (perdu à l'amorçage suivant). L'indicateur **–I** entraîne un changement différé à l'amorçage suivant.
Diagnostics :	**netstat –s** pour examiner les dépassements du tampon de socket UDP.
Optimisation :	Augmentez le nombre jusqu'à éliminer les dépassements.
Voir :	"Nombre de **biod** et de **nfsd** requis", page 9-36.

Décompte nfsd

Objet :	Nombre de processus **nfsd** disponibles pour gérer les requêtes NFS sur un serveur.
Valeurs :	Par défaut : Intervalle : 1 à *n*
Affichage :	**ps –ef** | **grep nfsd**
Modification :	**chnfs –n** *nouvelle-valeur.* Le changement prend effet immédiatement et est permanent. L'indicateur **–N** entraîne un changement immédiat, mais temporaire (perdu à l'amorçage suivant). L'indicateur **–I** entraîne un changement différé à l'amorçage suivant.
Diagnostics :	**netstat –s** pour examiner les dépassements du tampon de socket UDP.
Optimisation :	Augmentez le nombre jusqu'à éliminer les dépassements.
Voir :	"Nombre de biod et de nfsd requis", page 9-36.

dog_ticks

Objet :	Granularité de l'horloge pour les routines lfWatchdog. Cette valeur n'est pas utilisée sous AIX.
Valeurs :	Par défaut : 60
Affichage :	N/A.
Modification :	N/A.
Diagnostics :	N/A.
Optimisation :	N/A.
Voir :	N/A.

Intervalle de réessai fork()

Objet :	Délai d'attente avant relance d'un **fork** ayant échoué par manque d'espace de pagination.
Valeurs :	Par défaut : 10 (impulsions d'horloge de 10 millisecondes). Intervalle : 10 à *n* impulsions d'horloge.
Affichage :	**schedtune**
Modification :	**schedtune –f** *nouvelle-valeur.* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **schedtune** à /**etc**/**inittab**.
Diagnostics :	Si des processus ont été tués par manque d'espace de pagination, surveillez la situation à l'aide de la sous-routine **sigdanger()**.
Optimisation :	Si le manque d'espace de pagination est dû à des surcharges brèves et sporadiques, augmenter le delai de réessai peut entraîner un allongement du délai d'attente des processus permettant une libération d'espace de pagination. Sinon, il convient d'agrandir ces espaces.
Voir :	N/A.

Intervalle syncd

Objet :	Délai entre deux appels **sync()** par **syncd**.
Valeurs :	Par défaut : 60 (secondes). Intervalle : 1 à n'importe quel entier positif.
Affichage :	**grep syncd /sbin/rc.boot**
Modification :	**vi /sbin/rc.boot** Le changement est différé à l'amorçage suivant, mais permanent.
Diagnostics :	N/A.
Optimisation :	A sa valeur par défaut, l'effet de ce paramètre sur les performances est négligeable. Aucune modification conseillée. Une diminution significative de l'intervalle **syncd** – pour préserver l'intégrité des données peut avoir l'effet inverse.
Voir :	"Performances et sync/fsync", page 8-21.

ipforwarding

Objet :	Spécifie si le noyau fait suivre les paquets IP.
Valeurs :	Par défaut : 0 (non). Intervalle : 0 à 1
Affichage :	**no –a** ou **no –o ipforwarding**
Modification :	**no –o ipforwarding=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.
Diagnostics :	N/A.
Optimisation :	Il s'agit d'une décision de configuration ayant un impact sur les performances.
Voir :	N/A.

ipfragttl

Objet :	Durée de vie des fragments de paquets IP.
Valeurs :	Par défaut : 60 (demi-secondes). Intervalle : 60 à *n*
Affichage :	**no –a** ou **no –o ipfragttl**
Modification :	**no –o ipfragttl=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.
Diagnostics :	**netstat –s**
Optimisation :	Si la valeur de `IP: fragments dropped after timeout` est non nulle, augmenter **ipfragttl** peut réduire les retransmissions.
Voir :	N/A.

ipqmaxlen

Objet :	Nombre maximal d'entrées de la file d'entrée IP.
Valeurs :	Par défaut : Intervalle : 50 à *n*
Affichage :	**no –a** ou **no –o ipqmaxlen**
Modification :	**no –o ipqmaxlen =***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics :	Utilisez la commande **crash** pour accéder au compteur de dépassement de la file d'entrée IP.
Optimisation :	Augmentez la taille.
Voir :	"Optimisation des performances du protocole IP", page 9-23.

ipsendredirects

Objet :	Spécifie si le noyau envoie les signaux de réacheminement.
Valeurs :	Par défaut : 1 (oui). Intervalle : 0 à 1
Affichage :	**no –a** ou **no –o ipsendredirects**
Modification :	**no –o ipsendredirects=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics :	N/A.
Optimisation :	N/A. Il s'agit d'une décision de configuration ayant un impact sur les performances.
Voir :	N/A.

Limites des requêtes en sortie sur une carte disque

Objet :	Nombre maximal de requêtes en sortie sur un bus SCSI (ne s'applique qu'à la carte SCSI–2 Fast/Wide).
Valeurs :	Par défaut : Intervalle : 40 à 128
Affichage :	**lsattr –E –l scsi***n* **–a num_cmd_elems**
Modification :	**chdev –l scsi***n* **–a num_cmd_elems=***nouvellevaleur i* Le changement prend effet immédiatement et est permanent. L'indicateur **–T** entraîne un changement immédiat, mais perdu à l'amorçage suivant. L'indicateur **–P** entraîne un changement différé à l'amorçage suivant, mais permanent.
Diagnostics :	N/A.
Optimisation :	La valeur doit être égale au nombre d'unités physiques (y compris celles se trouvant dans des piles de disques) situées sur le bus SCSI, multiplié par la longueur des files d'attente sur chaque unité.
Voir :	"Limites de la file d'attente de l'unité de disque et de la carte SCSI", page 8-23.

Longueur de file d'attente d'unité disque

Objet :	Nombre maximal de requêtes pouvant se trouver dans la file d'attente de l'unité de disque.
Valeurs :	Par défaut : disques BULL = 3. Intervalle : N/A. Par défaut : disques non BULL = 0. Intervalle : spécifié par le fabricant.
Affichage :	**lsattr –E –l hdisk***n.*
Modification :	**chdev –l hdisk***n* **–a q_type=simple –a queue_depth=***nouvelle-valeur.* Le changement prend effet immédiatement et est permanent. L'indicateur **–T** entraîne un changement immédiat, mais perdu à l'amorçage suivant. L'indicateur **–P** entraîne un changement différé à l'amorçage suivant, mais permanent.
Diagnostics :	N/A.
Optimisation :	Si le disque non BULL gère la mise des requêtes en file d'attente, ce changement doit être effectué pour garantir que le système exploite bien cette capacité.
Voir :	"Limites de la file d'attente de l'unité de disque et de la carte SCSI", page 8-23.

loop_check_sum (version 3.2.5 seulement)

Objet :	Spécifie si les totaux de contrôle sont effectués et vérifiés sur une interface en boucle. (Cette fonction n'existe pas sous AIX version 4.1.)
Valeurs :	Par défaut : 1 (oui). Intervalle : 0 à 1
Affichage :	**no –a** ou **no –o loop_check_sum**
Modification :	**no –o loop_check_sum=0** La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.
Diagnostics :	N/A.
Optimisation :	Nous vous conseillons de désactiver la vérification des totaux de contrôle (**loop_check_sum=0**).
Voir :	N/A.

lowclust (version 3.2.5 seulement)

Objet :	Spécifie le niveau bas du pool de grappes mbuf.
Valeurs :	Par défaut : dépend de la configuration. Intervalle : 5 à *n*
Affichage :	**no –a** ou **no –o lowclust**
Modification :	**no –o lowclust=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.
Diagnostics :	**netstat –m**
Optimisation :	Si "requests for memory denied" est non nul, augmentez **lowclust**.
Voir :	"Optimisation du pool mbuf", page 9-26.

lowmbuf (version 3.2.5 seulement)

Objet :	Spécifie le niveau bas du pool mbuf.
Valeurs :	Par défaut : dépend de la configuration. Intervalle : 64 à *n*
Affichage :	**no –a** ou **no –o lowmbuf**
Modification :	**no –o lowmbuf=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics :	**netstat –m**
Optimisation :	Si "requests for memory denied" est non nul, augmentez **lowmbuf**.
Voir :	"Optimisation du pool mbuf", page 9-26.

lvm_bufcnt (AIX version 4.1 seulement)

Objet :	Nombre de tampons LVM pour les E/S physiques brutes.
Valeurs :	Par défaut : Intervalle : 1 à 64
Affichage :	**vmtune**
Modification :	**vmtune –u** *nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **vmtune** à /**etc**/**inittab**.
Diagnostics :	Les applications qui effectuent de longues écritures sur des volumes logiques bruts répartis n'atteignent pas le débit souhaité.
Optimisation :	Si un système est configuré avec des volumes logiques bruts répartis et effectue des écritures de plus de 1,125 Mo, augmenter cette valeur peut améliorer le débit de l'application.
Voir :	Commande **vmtune"**, page A-9.

maxrandwrt (AIX version 4.1.3 et ultérieure)

Objet :	Nombre de pages de fichier sales à accumuler en RAM avant qu'elles soient synchronisées sur disque via un algorithme d'écriture différée. Le seuil d'écriture différée aléatoire est défini sur la base des fichiers.
Valeurs :	Par défaut : Intervalle : 0 à 128 (pages de 4 ko).
Affichage :	**vmtune**
Modification :	**vmtune –W** *nouvelle-valeur.* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **vmtune** à /**etc**/**inittab**.
Diagnostics :	**vmstat** *n* affiche les manques de page et les pointes d'attente d'E/S à intervalles réguliers (généralement lorsque le démon **sync** écrit des pages sur disque).
Optimisation :	Si **vmstat** *n* affiche des manques de page et des pointes d'attente d'E/S à intervalles réguliers (généralement lorsque le démon **sync** écrit des pages sur le disque), régler la valeur de **maxrandwrt** aide à mieux répartir les E/S. La valeur 0 désactive l'écriture aléatoire différée.
Voir :	"Gestion AIX du stockage sur disques fixes", page 2-13 et la **commande vmtune**, page A-9.

maxbuf

Objet :	Nombre de pages (4 ko) dans le cache tampon d'E/S bloc.
Valeurs :	Par défaut : Intervalle : x à y
Affichage :	**lsattr –E –l sys0 –a maxbuf**
Modification :	**chdev –l sys0 –a maxbuf=***nouvelle-valeur.* Le changement prend effet immédiatement et est permanent. L'indicateur **–T** entraîne un changement immédiat, mais perdu à l'amorçage suivant. L'indicateur **–P** entraîne un changement différé à l'amorçage suivant, mais permanent.
Diagnostics :	N/A.
Optimisation :	Ce paramètre a normalement un effet négligeable sur les performances d'un système AIX, dans la mesure où les E/S ordinaires ne passent pas par le cache tampon des E/S bloc.
Voir :	N/A.

max_coalesce

Objet :	Spécifie la taille maximale, en octets, des requêtes fusionnées par le pilote d'unité SCSI dans sa file d'attente.
Valeurs :	Par défaut : ko. Intervalle : 64 ko à 2 Go
Affichage :	**odmget**
Modification :	**odmdelete**, **odmadd**, **bosboot** Le changement est différé à l'amorçage suivant, mais permanent.
Diagnostics :	N/A.
Optimisation :	Augmentez si des volumes logiques répartis ou des piles de disques sont utilisés.
Voir :	"Modification du paramètre **max_coalesce** du pilote SCSI", page 8-22.

maxfree

Objet :	Taille maximale à partir de laquelle la liste de trames de page disponibles VMM croît – par vol de page.
Valeurs :	Par défaut : dépend de la configuration. Intervalle : 16 à 204800 (trames de 4 ko).
Affichage :	**vmtune**
Modification :	**vmtune –F** *nouvelle-valeur.* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **vmtune** à /**etc**/**inittab**.
Diagnostics :	Examinez les changements intervenus au niveau de la taille de la liste des disponibilités, via **vmstat** *n*.
Optimisation :	Si **vmstat** *n* indique que cette taille passe fréquemment en dessous de **minfree** par le fait de demandes émises par des applications, augmentez **maxfree** de façon à réduire les appels à la liste de disponibilités remplie. D'une façon générale, maintenez **maxfree** – **minfree** <= 100.
Voir :	"Optimisation du remplacement de page VMM", page 7-16.

maxperm

Objet :	Pourcentage de trames de page mémoire occupées par des pages permanentes, au-delà duquel seules les pages permanentes voient leur trames volées.
Valeurs :	Par défaut : % de (taille mémoire – 4 Mo). Intervalle : 5 à 100
Affichage :	**vmtune**
Modification :	**vmtune –P** *nouvelle-valeur.* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **vmtune** à **/etc/inittab**.
Diagnostics :	Surveillez les E/S disque via **iostat** *n*.
Optimisation :	Si vous savez que des fichiers sont lus répétitivement, mais que les taux d'E/S ne baissent pas depuis le lancement, **maxperm** est sans doute trop bas.
Voir :	"Optimisation du remplacement de page VMM", page 7-16.

maxpgahead

Objet :	Limite supérieure du nombre de pages lues par anticipation par VMM lors du traitement d'un fichier à accès séquentiel.
Valeurs :	Par défaut : Intervalle : 0 à 16
Affichage :	**vmtune**
Modification :	**vmtune –R** *nouvelle-valeur.* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **vmtune** à **/etc/inittab**.
Diagnostics :	Observez la durée d'exécution des applications critiques au niveau des E/S séquentielles, via la commande **time**.
Optimisation :	Si la durée d'exécution diminue lorsque **maxpgahead** est plus élevé, étudiez les autres applications pour vérifier que leurs performances n'ont pas souffert.
Voir :	"Optimisation des lectures séquentielles anticipées", page 8-11.

maxpin (AIX version 4.1 seulement)

Objet :	Pourcentage maximal de mémoire réelle susceptible d'être fixée.
Valeurs :	Par défaut : 80 (% de RAM). Intervalle : au moins 4 Mo fixable à au moins 4 Mo non fixable.
Affichage :	**vmtune**
Modification :	**vmtune –M** *nouvelle-valeur.* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant.
Diagnostics :	N/A.
Optimisation :	N'effectuez de modification que dans des cas extrêmes, tels que l'évaluation de la charge maximale.
Voir :	Commande **vmtune**, page A-9.

maxpout

Objet :	Spécifie le nombre maximal d'E/S en attente d'un fichier.
Valeurs :	Par défaut : 0 (pas de contrôle). Intervalle : 0 à *n* (*n* étant un multiple de 4, augmenté de 1).
Affichage :	**lsattr –E –l sys0 –a maxpout**
Modification :	**chdev –l sys0 –a maxpout=***nouvelle-valeur.* Le changement prend effet immédiatement et est permanent. L'indicateur **–T** entraîne un changement immédiat, mais perdu à l'amorçage suivant. L'indicateur **–P** entraîne un changement différé à l'amorçage suivant, mais permanent.
Diagnostics :	Si les temps de réponse en avant-plan de programmes générant de lourdes sorties séquentielles sur disque se dégradent, il convient sans doute de réguler les sorties séquentielles.
Optimisation :	Définissez **maxpout** à 33 et **minpout** à 16. Si les performances en mode séquentiel se dégradent de façon inacceptable, augmentez l'un ou l'autre de ces paramètres, ou les deux. Si les performances en avant-plan restent inacceptables, diminuez les deux.
Voir :	"Régulation des E/S disque", page 8-13.

maxttl

Objet :	Durée de vie des paquets RIP (Routing Information Protocol).
Valeurs :	Par défaut : Intervalle : N/A.
Affichage :	**no –a** or **no –o maxttl**
Modification :	**no –o maxttl=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.
Diagnostics :	N/A.
Optimisation :	N/A.
Voir :	N/A.

mb_cl_hiwat (version 3.2.5 seulement)

Objet :	Spécifie le niveau haut du pool de grappes mbuf.
Valeurs :	Par défaut : dépend de la configuration. Intervalle : N/A.
Affichage :	**no –a** or **no –o mb_cl_hiwat**
Modification :	**no –o mb_cl_hiwat=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.
Diagnostics :	**netstat –m**
Optimisation :	Si le nombre de grappes mbuf (appelées "pages mappées" par **netstat**) est régulièrement supérieur à **mb_cl_hiwat**, augmentez **mb_cl_hiwat**.
Voir :	"Optimisation du pool mbuf", page 9-26.

minfree

Objet :	Taille de la liste de trames de page VMM disponibles à partir de laquelle commence à voler des pages pour remplir la liste de disponibilités. Valeurs :
Valeurs :	Par défaut : dépend de la configuration. Intervalle : x à n'importe quel entier positif.
Affichage :	**vmtune**
Modification :	**vmtune –f** *nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **vmtune** à /**etc**/**inittab**.
Diagnostics :	**vmstat** n.
Optimisation :	Si des processus sont différés par le vol de page, augmentez **minfree** pour améliorer les temps de réponse. Augmentez **maxfree** de la même valeur ou d'une valeur supérieure.
Voir :	"Optimisation du remplacement de page VMM", page 7-16.

minperm

Objet :	Pourcentage de trames de page occupées par des pages permanentes, en deçà duquel VMM vole les trames des pages permanentes et des pages de travail, quel que soit le taux de repages.
Valeurs :	Par défaut : % de (taille mémoire – 4 Mo). Intervalle : 5 à 100
Affichage :	**vmtune**
Modification :	**vmtune –P** *nouvelle-valeur.* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **vmtune** à /**etc**/**inittab**.
Diagnostics :	Surveillez les E/S disque via **iostat** *n*.
Optimisation :	Si vous savez que des fichiers sont lus répétitivement, mais que les taux d'E/S ne baissent pas depuis le lancement, **minperm** est sans doute trop bas.
Voir :	"Optimisation du remplacement de page VMM", page 7-16.

minpgahead

Objet :	Nombre de pages lues par anticipation par VMM dès qu'il détecte un accès séquentiel.
Valeurs :	Par défaut : Intervalle : 0 à 16
Affichage :	**vmtune**
Modification :	**vmtune –r** *nouvelle-valeur.* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **vmtune** à /**etc**/**inittab**.
Diagnostics :	Observez la durée d'exécution des applications critiques au niveau des E/S séquentielles, via la commande **time**.
Optimisation :	Si la durée d'exécution diminue lorsque **minpgahead** est plus élevé, étudiez les autres applications pour vérifier que leurs performances n'ont pas souffert.
Voir :	"Optimisation des lectures séquentielles anticipées", page 8-11.

minpout

Objet :	Spécifie le point à partir duquel les programmes ayant atteint **maxpout** peuvent reprendre l'écriture dans le fichier.
Valeurs :	Par défaut : 0 (pas de contrôle). Intervalle : 0 à *n* (*n* étant un multiple de 4 et inférieur d'au moins 4 à **maxpout**).
Affichage :	**lsattr –E –l sys0 –a minpout**
Modification :	**chdev –l sys0 –a minpout=***nouvelle-valeur.* Le changement prend effet immédiatement et est permanent. L'indicateur **–T** entraîne un changement immédiat, mais perdu à l'amorçage suivant. L'indicateur **–P** entraîne un changement différé à l'amorçage suivant, mais permanent.
Diagnostics :	Si les temps de réponse en avant-plan de programmes générant de lourdes sorties séquentielles sur disque se dégradent, il convient sans doute de réguler les sorties séquentielles.
Optimisation :	Définissez **maxpout** à 33 et **minpout** à 16. Si les performances en mode séquentiel se dégradent de façon inacceptable, augmentez l'un ou l'autre de ces paramètres, ou les deux. Si les performances en avant-plan restent inacceptables, diminuez les deux.
Voir :	"Régulation des E/S disque", page 8-13.

MTU

Objet :	Limite la taille des paquets transmis via le réseau.
Valeurs :	**tr***n* (4 Mo) : Par défaut : 1492, Intervalle : 60 à 3900 **tr***n* (16 Mo) : Par défaut : 1492, Intervalle : 60 à 17960 **en***n*: Par défaut : 1500, Intervalle : 60 à 1500 **fi***n*: Par défaut : 4352, Intervalle : 60 à 4352 **hi***n*: Par défaut : 65536, Intervalle : 60 à 65536 **so***n*: Par défaut : 61428, Intervalle : 60 à 61428 **lo***n*: Par défaut : 1500 (version 3.2.5) 16896 (AIX version 4.1), Intervalle : 60 à 65536
Affichage :	**lsattr –E –l tr***n*
Modification :	**chdev –l tr***n* **–a mtu=***nouvelle-valeur.* Ne peut être modifié en cours d'exploitation de l'interface. Tous les systèmes du réseau local (LAN) devant être dotés du même MTU, ils doivent être modifiés simultanément. Le changement reste effectif même après réamorçage.
Diagnostics :	Statistiques de fragmentation de paquets
Optimisation :	Augmentez la taille MTU pour les interfaces en anneau à jeton : **tr***n* (4 Mo) : 4056 **tr***n* (16 Mo) : 8500 Pour l'interface en boucle **lo***n* sous la version 3.2.5, augmentez à 16896. Pour les autres interfaces, conservez la valeur par défaut.
Voir :	"Cartes réseau local et pilotes d'unités", page 9-11.

nfs_chars (version 3.2.5), nfs_socketsize (AIX version 4.1)

Objet :	Taille du tampon de socket UDP NFS.
Valeurs :	Par défaut : Intervalle : 60000 à (**sb_max** –128).
Affichage :	**nfso –a** ou **nfso –o nfs_chars** (sous AIX version 4.1, **nfso –o nfs_socketsize**).
Modification :	**nfso –o nfs_chars=***nouvelle-valeur*. (sous AIX version 4.1, **nfso –o nfs_socketsize=***nouvelle-valeur*) **stopsrc –g nfs.** **startsrc –g nfs** La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **nfso** à /**etc**/**rc.nfs** ou /**etc**/**rc.net**. **sb_max** doit d'abord être modifié en conséquence.
Diagnostics :	**netstat –s**
Optimisation :	Si le compte "UDP: socket buffer overflows" est non nul, augmentez **sb_max** et **nfs_chars**.
Voir :	"Optimisation de NFS", page 9-36.

nfs_gather_threshold (AIX version 4.1 seulement)

Objet :	Taille minimale d'une écriture en "veille" avant synchronisation. Utilisé pour désactiver la fragmentation/assemblage des écritures sur le même vnode.
Valeurs :	Par défaut : Intervalle : x à y
Affichage :	**nfso –a** ou **nfso –o nfs_gather_threshold**
Modification :	**nfso –o nfs_gather_threshold=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant.
Diagnostics :	N/A.
Optimisation :	N/A.
Voir :	N/A.

nfs_portmon (version 3.2.5), portcheck (AIX version 4.1)

Objet :	Spécifie si NFS opère un contrôle, que les requêtes soient issues ou non de ports privilégiés.
Valeurs :	Par défaut : 0 (non). Intervalle : 0 à 1
Affichage :	**nfso –a** ou **nfso –o nfs_portmon**
Modification :	**nfso –o nfs_portmon=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **nfso** à /**etc**/**rc.nfs**.
Diagnostics :	N/A.
Optimisation :	Il s'agit d'une décision de configuration ayant un impact minimal sur les performances.
Voir :	N/A.

nfs_repeat_messages (AIX version 4.1 seulement)

Objet :	Spécifie si les messages écrits par NFS doivent être répétés.
Valeurs :	Par défaut : 1 (oui). Intervalle : 0 à 1
Affichage :	**nfso –a** ou **nfso –o nfs_repeat_messages**
Modification :	**nfso –o nfs_repeat_messages=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant.
Diagnostics :	N/A.
Optimisation :	N/A.
Voir :	N/A.

nfs_setattr_error (AIX version 4.1 seulement)

Objet :	Spécifie si NFS ignore les erreurs NFS dues à des définitions d'attributs PC illégaux.
Valeurs :	Par défaut : Intervalle : 0 à 1
Affichage :	**nfso –a**
Modification :	**nfso –o nfs_setattr_error=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant.
Diagnostics :	N/A.
Optimisation :	N/A.
Voir :	N/A.

nfsudpcksum (version 3.2.5), udpchecksum (AIX version 4.1)

Objet :	Spécifie si NFS effectue le total de contrôle UDP.
Valeurs :	Par défaut : 1 (oui). Intervalle : 0 à 1
Affichage :	**nfso –a** or **nfso –o nfsudpcksum**
Modification :	**nfso –o nfsudpcksum=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **nfso** à /**etc**/**rc.nfs**.
Diagnostics :	N/A.
Optimisation :	Désactiver le calcul du total de contrôle peut réduire le temps de traitement, mais augmente le risque d'erreurs de données non détectées.
Voir :	N/A.

nonlocsrcroute

Objet :	Indique que les paquets IP acheminés strictement depuis la source peuvent être adressés à des hôtes hors de l'anneau local. (Le routage depuis une source non stricte n'est pas affecté.)
Valeurs :	Par défaut : 0 (non). Intervalle : 0 à 1
Affichage :	**no –a** or **no –o nonlocsrcroute**
Modification :	**no –o nonlocsrcroute=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.
Diagnostics :	N/A.
Optimisation :	Il s'agit d'une décision de configuration ayant un impact minimal sur les performances.
Voir :	N/A.

npskill (AIX version 4.1 seulement)

Objet :	Nombre de pages libres de l'espace de pagination, à partir duquel les processus commencent à être tués.
Valeurs :	Par défaut : Intervalle : 0 au nombre de pages en mémoire réelle.
Affichage :	**vmtune**
Modification :	**vmtune –k** *nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant.
Diagnostics :	N/A.
Optimisation :	N/A.
Voir :	Commande **vmtune**, page A-9.

npswarn (AIX version 4.1 seulement)

Objet :	Nombre de pages libres de l'espace de pagination, à partir duquel les processus commencent à recevoir SIGDANGER.
Valeurs :	Par défaut : Intervalle : au moins **npskill** pour le nombre de pages en mémoire réelle.
Affichage :	**vmtune**
Modification :	**vmtune –w** *nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant.
Diagnostics :	N/A.
Optimisation :	Augmentez si vous constatez que des processus ont été tués par insuffisance d'espace de pagination.
Voir :	Commande **vmtune**, page A-9.

numclust (AIX version 4.1 seulement)

Objet :	Nombre de grappes de 16 ko traitées en écriture différée.
Valeurs :	Par défaut : Intervalle : 1 à n'importe quel entier positif.
Affichage :	**vmtune**
Modification :	**vmtune –c** *nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant.
Diagnostics :	N/A.
Optimisation :	Augmentez éventuellement si des volumes logiques répartis ou des piles de disques sont utilisés.
Voir :	Commande **vmtune**, page A-9.

numfsbuf (AIX version 4.1 seulement)

Objet :	Nombre de systèmes de fichiers `bufstruct`.
Valeurs :	Par défaut : Intervalle : 64 à n'importe quel entier positif.
Affichage :	**vmtune**
Modification :	**vmtune –b** *nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant.
Diagnostics :	N/A.
Optimisation :	Augmentez éventuellement si des volumes logiques répartis ou des piles de disques sont utilisés.
Voir :	Commande **vmtune**, page A-9.

Paramètres de contrôle de charge mémoire

Objet :	Personnalise l'utilitaire de contrôle de la charge mémoire VMM pour maximiser l'utilisation du système tout en évitant l'emballement. Les paramètres les plus utilisés sont : *h* seuil de surcharge de la mémoire haute. *p* seuil de surcharge mémoire par les processus. *m* niveau de multiprogrammation minimal.
Valeurs :	*h* Valeur par défaut : Intervalle : 0 à n'importe quel entier positif. *p* Valeur par défaut : Intervalle : 0 à n'importe quel entier positif. *m* Valeur par défaut : Intervalle : 0 à n'importe quel entier positif.
Affichage :	**schedtune**
Modification :	**schedtune** [**–h** *nouvelle-valeur*] [**–p** *nouvelle-valeur*] [**–m** *nouvelle-valeur*] La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **schedtune** à **/etc/inittab**.
Diagnostics :	De lourdes charges mémoire peuvent induire d'importants écarts au niveau des temps de réponse.

Optimisation : **schedtune –h 0** désactive le contrôle de la charge mémoire.
schedtune –p 2 requiert un niveau plus élevé de repagination par un processus donné, avant qu'il ne soit candidat à être suspendu par le contrôle de charge mémoire.
schedtune –m 10 requiert que le contrôle de charge mémoire maintienne au moins 10 processus utilisateur actifs lorsqu'il suspend des processus.

Voir : "Utilitaire de contrôle de charge mémoire", page 2-8 et "Optimisation du contrôle de charge mémoire VMM", page 7-14.

pd_npages

Objet : Nombre de pages à supprimer d'un coup de la RAM lorsqu'un fichier est supprimé.

Valeurs : Valeur par défaut :*taille du plus grand fichier* / *taille de page*. Intervalle : 1 à *taille du plus grand fichier* / *taille de page*

Affichage : **vmtune**

Modification : **vmtune –N** *nouvelle-valeur*
La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **vmtune** à **/etc/inittab**.

Diagnostics : Cette option est utile si une application en temps réel voit son temps de réponse singulièrement ralenti pendant la suppression de gros fichiers.

Optimisation : Si le temps réel de réponse est un élément critique, régler cette option peut l'améliorer en répartissant la suppression des pages de fichier de la RAM plus uniformément par rapport à la charge de travail.

Voir : Commande **vmtune"**, page A-9.

rec_que_size

Objet : (optimisable sous AIX version 3 exclusivement). Spécifie le nombre maximal de tampons de réception qui peuvent être en file d'attente pour l'interface.

Valeurs : Par défaut : Intervalle : 20 à 150

Affichage : **lsattr –E –l tok***n* **–a rec_que_size**

Modification : **ifconfig tr0 detach**
chdev –l tok*n* **–a rec_que_size=***nouvelle-valeur.*
ifconfig tr0 *nom-hôte* **up**
Le changement reste effectif même après réamorçage.

Diagnostics : N/A.

Optimisation : Augmentez la taille. Doit être naturellement définie à 150 sur les systèmes orientés réseau, et notamment les serveurs.

Voir : "Cartes réseau local et pilotes d'unités", page 9-11.

rfc1122addrchk

Objet : Spécifie si la validation d'adresse est effectuée entre les couches de communication.

Valeurs : Par défaut : 0 (non). Intervalle : 0 à 1

Affichage : **no –a** ou **no –o rfc1122addrchk**

Modification : **no –o rfc1122addrchk=***nouvelle-valeur*
La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.

Diagnostics : N/A.
Optimisation : Cette valeur ne doit pas être modifiée.
Voir : N/A.

rfc1323

Objet : 1 indique que **tcp_sendspace** et **tcp_recvspace** peuvent dépasser 64 ko.
Valeurs : Par défaut : Intervalle : 0 ou 1
Affichage : **no –a** ou **no –o rfc1323**
Modification : **no –o rfc1323=***nouvelle-valeur*
La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics : Aucun.
Optimisation : Effectuez la modification avant de tenter de donner à **tcp_sendspace** et **tcp_recvspace** une valeur supérieure à 64 ko.
Voir : "Couche TCP", page 9-6.

sb_max

Objet : Limite supérieure absolue de la taille des tampons de socket TCP et UDP. Limite **setsockopt()**, **udp_sendspace, udp_recvspace**, **tcp_sendspace** et **tcp_recvspace**.
Valeurs : Par défaut : Intervalle : N/A.
Affichage : **no –a** ou **no –o sb_max**
Modification : **no –o sb_max=***nouvelle-valeur*
La modification est immédiate pour les nouvelles connexions. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics : Aucun.
Optimisation : Augmentez la taille, de préférence à un multiple de 4 096. Doit être environ double de la limite du plus grand tampon de socket.
Voir : "Couche socket", page 9-3.

subnetsarelocal

Objet : Spécifie que tous les sous-réseaux conformes à ce masque doivent être considérés comme locaux, pour des opérations telles que l'établissement, par exemple, de la taille maximale de segment TCP.
Valeurs : Par défaut : 1 (oui). Intervalle : 0 à 1
Affichage : **no –a** ou **no –o subnetsarelocal**
Modification : **no –o subnetsarelocal=***nouvelle-valeur*
La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics : N/A.

Optimisation : Il s'agit d'une décision de configuration ayant un impact sur les performances. Si les sous-réseaux n'ont pas tous le même MTU, la fragmentation au niveau des ponts peut entraîner une dégradation des performances. S'ils ont tous le même MTU, et que **subnetsarelocal** vaut 0, les sessions TCP risquent d'utiliser inutilement un MSS trop petit.

Voir : "Optimisation de la taille maximum de segment (MSS) TCP", page 9-21.

Taille de l'espace de pagination

Objet : Quantité d'espace disque requise pour les pages de travail.

Valeurs : Par défaut : dépend de la configuration. Intervalle : 32 Mo à *n* Mo pour hd6, 16 Mo à *n* Mo pour non hd6

Affichage : **lsps –a**

Modification : **mkps** ou **chps** ou **smit pgsp**
La modification est immédiate et est permanente. L'espace de pagination n'est toutefois pas forcément mis en service immédiatement.

Diagnostics : **lsps –a.** Si des processus ont été tués par insuffisance d'espace de pagination, surveillez la situation à l'aide de la sous-routine **psdanger()**.

Optimisation : S'il apparaît que l'espace de pagination est insuffisant pour gérer la charge de travail normale, ajoutez un nouvel espace sur un autre volume physique ou agrandissez l'espace existant.

Voir : "Position et taille des espaces de pagination", page 4-26.

tcp_keepidle

Objet : Délai total pendant lequel conserver active une connexion TCP inoccupée.

Valeurs : Par défaut : 14400 (demi-secondes) = 2 heures. Intervalle : tout entier positif.

Affichage : **no –a** ou **no –o tcp_keepidle**

Modification : **no –o tcp_keepidle=***nouvelle-valeur*
La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.

Diagnostics : N/A.

Optimisation : Il s'agit d'une décision de configuration ayant un impact minimal sur les performances. Aucune modification conseillée.

Voir : N/A.

tcp_keepintvl

Objet : Intervalle entre les paquets envoyés pour valider la connexion TCP.

Valeurs : Par défaut : 150 (demi-secondes) = 75 secondes. Intervalle : tout entier positif.

Affichage : **no –a** ou **no –o tcp_keepintvl**

Modification : **no –o tcp_keepintvl=***nouvelle-valeur*
La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.

Diagnostics : N/A.

Optimisation : Il s'agit d'une décision de configuration ayant un impact minimal sur les performances. Aucune modification conseillée. Si l'intervalle est sérieusement raccourci, les coûts en matière de traitement et de largeur de bande peuvent devenir significatifs.
Voir : N/A.

tcp_mssdflt

Objet : Taille de segment maximale utilisée par défaut pour communiquer avec les réseaux distants.
Valeurs : Par défaut : Intervalle : 512 à (MTU de réseau local – 64).
Affichage : **no –a** ou **no –o tcp_mssdflt**
Modification : **no –o tcp_mssdflt=***nouvelle-valeur*
La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics : N/A.
Optimisation : Augmentez, si vous le souhaitez.
Voir : "Optimisation de la taille maximum de segment (MSS) TCP", page 9-21.

tcp_recvspace

Objet : Valeur par défaut de la taille du tampon de réception du socket TCP.
Valeurs : Par défaut : Intervalle : 0 à 64 ko si **rfc1323=0,**
Intervalle : 0 à 4 Go si `rfc1323=1`.
Doit être inférieur ou égal à **sb_max**.
Doit être égal à **tcp_sendspace** et uniforme sur tous les systèmes AIX fréquemment sollicités.
Affichage : **no –a** ou **no –o tcp_recvspace**
Modification : **no –o tcp_recvspace=***nouvelle-valeur*
La modification est immédiate pour les nouvelles connexions. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics : Débit faible
Optimisation : Augmentez la taille, de préférence à un multiple de 4 096.
Voir : "Couche socket", page 9-3.

tcp_sendspace

Objet : Valeur par défaut de la taille du tampon d'envoi du socket TCP.
Valeurs : Par défaut : Intervalle : 0 à 64 ko si **rfc1323=0,**
Intervalle : 0 à 4 Go si **rfc1323=1**.
Doit être inférieur ou égal à **sb_max**.
Doit être égal à **tcp_recvspace** et uniforme sur tous les systèmes AIX fréquemment sollicités.
Affichage : **no –a** ou **no –o tcp_sendspace**
Modification : **no –o tcp_sendspace=***nouvelle-valeur*
La modification est immédiate pour les nouvelles connexions. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics : Débit faible
Optimisation : Augmentez la taille, de préférence à un multiple de 4 096.
Voir : "Couche socket", page 9-3.

tcp_ttl

Objet :	Durée de vie des paquets TCP.
Valeurs :	Par défaut : 60 (impulsions de processeur de 10 millisecondes). tout entier positif.
Affichage :	Affichage : **no –a** ou **no –o tcp_ttl**
Modification :	**no –o tcp_ttl=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics :	**netstat –s**
Optimisation :	Si le système subit des dépassements de délai TCP, augmenter **tcp_ttl** peut réduire le nombre de retransmissions.
Voir :	N/A.

thewall

Objet :	Limite supérieure absolue de la quantité de mémoire réelle susceptible d'être utilisée par le sous-système de communication.
Valeurs :	Par défaut : % de la mémoire réelle. Intervalle : 0 à 50 % de la mémoire réelle.
Affichage :	**no –a** ou **no –o thewall**
Modification :	**no –o thewall=***nouvelle-valeur* *nouvelle-valeur* est en ko, et non en octets. La modification est immédiate pour les nouvelles connexions. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics :	Aucun.
Optimisation :	Augmentez la taille, de préférence à un multiple de 4 ko.
Voir :	"Optimisation du pool mbuf", page 9-26.

udp_recvspace

Objet :	Valeur par défaut de la taille du tampon de réception du socket UDP.
Valeurs :	Par défaut : Intervalle : N/A. Doit être inférieur ou égal à **sb_max**.
Affichage :	**no –a** ou **no –o udp_recvspace**
Modification :	**no –o udp_recvspace=***nouvelle-valeur* La modification est immédiate pour les nouvelles connexions. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à /**etc**/**rc.net**.
Diagnostics :	*n* non nul dans le compte rendu de **netstat –s** de **udp:** *n* **socket buffer overflows**
Optimisation :	Augmentez la taille, de préférence à un multiple de 4 096.
Voir :	"Couche socket", page 9-3.

udp_sendspace

Objet :	Valeur par défaut de la taille du tampon d'envoi du socket UDP.
Valeurs :	Par défaut : Intervalle : 0 à 65536 Doit être inférieur ou égal à **sb_max**.
Affichage :	**no –a** ou **no –o udp_sendspace**
Modification :	**no –o udp_sendspace=***nouvelle-valeur* La modification est immédiate pour les nouvelles connexions. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.
Diagnostics :	N/A.
Optimisation :	Augmentez la taille, de préférence à un multiple de 4 096.
Voir :	"Couche socket", page 9-3.

udp_ttl

Objet :	Durée de vie des paquets UDP.
Valeurs :	Par défaut : 30 (impulsions d'horloge de 10 millisecondes). Intervalle : tout entier positif.
Affichage :	**no –a** ou **no –o udp_ttl**
Modification :	**no –o udp_ttl=***nouvelle-valeur* La modification est immédiate. La modification est effective jusqu'à l'amorçage système suivant. Pour un changement permanent, ajoutez la commande **no** à **/etc/rc.net**.
Diagnostics :	N/A.
Optimisation :	N/A.
Voir :	N/A.

xmt_que_size

Objet :	Spécifie le nombre maximal de tampons d'envoi susceptibles d'être en file d'attente pour l'unité.
Valeurs :	Par défaut : Intervalle : 20 à 150
Affichage :	**lsattr –E –l tok0 –a xmt_que_size**
Modification :	**ifconfig tr0 detach** **chdev –l tok0 –a xmt_que_size=***nouvelle-valeur* **ifconfig tr0** *nom-hôte* **up** Le changement reste effectif même après réamorçage.
Diagnostics :	**netstat –i** `Oerr > 0`
Optimisation :	Augmentez la taille. Doit être naturellement définie à 150 sur les systèmes orientés réseau, et notamment les serveurs.
Voir :	"Cartes réseau local et pilotes d'unités", page 9-11.

Index

Symboles

A

B

C

D

E

F

G

H

I

L

M

N

S

T

U

V

X

Vos remarques sur ce document / Technical publication remark form

Titre / Title : Bull AIX 4.3 Guide d'optimisation

Nº Référence / Reference Nº : 86 F2 72AP 04

Daté / Dated : Octobre 1999

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront examinées attentivement.
Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please furnish your complete mailing address below.

NOM / NAME : _____ Date : _____

SOCIETE / COMPANY : _____

ADRESSE / ADDRESS : _____

_____

Remettez cet imprimé à un responsable BULL ou envoyez-le directement à :

Please give this technical publication remark form to your BULL representative or mail to:

BULL ELECTRONICS ANGERS
CEDOC
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

Technical Publications Ordering Form
Bon de Commande de Documents Techniques

To order additional publications, please fill up a copy of this form and send it via mail to:
Pour commander des documents techniques, remplissez une copie de ce formulaire et envoyez-la à :

BULL ELECTRONICS ANGERS
CEDOC
ATTN / MME DUMOULIN
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

Managers / Gestionnaires :
Mrs. / Mme : **C. DUMOULIN** +33 (0) 2 41 73 76 65
Mr. / M : **L. CHERUBIN** +33 (0) 2 41 73 63 96

FAX : +33 (0) 2 41 73 60 19
E–Mail / Courrier Electronique : srv.Cedoc@franp.bull.fr

Or visit our web site at: / Ou visitez notre site web à:
`http://www-frec.bull.com` (PUBLICATIONS, Technical Literature, Ordering Form)

CEDOC Reference # Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	

[_ _] : **no revision number means latest revision** / pas de numéro de révision signifie révision la plus récente

NOM / NAME : ____________________ Date : __________

SOCIETE / COMPANY : ____________________

ADRESSE / ADDRESS : ____________________

____

TELEPHONE / PHONE : ____________________ FAX : ____________________

E–MAIL : ____________________

For Bull Subsidiaries / Pour les Filiales Bull :

Identification: ____________________

For Bull Affiliated Customers / Pour les Clients Affiliés Bull :

Customer Code / Code Client : ____________________

For Bull Internal Customers / Pour les Clients Internes Bull :

Budgetary Section / Section Budgétaire : ____________________

For Others / Pour les Autres :

Please ask your Bull representative. / Merci de demander à votre contact Bull.

PLACE BAR CODE IN LOWER LEFT CORNER

BULL ELECTRONICS ANGERS
CEDOC
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

86 F2 72AP 04

Bull

Utiliser les marques de découpe pour obtenir les étiquettes.
Use the cut marks to get the labels.

AIX
AIX 4.3
Guide
d'optimisation

86 F2 72AP 04

AIX
AIX 4.3
Guide
d'optimisation

86 F2 72AP 04

AIX
AIX 4.3
Guide
d'optimisation

86 F2 72AP 04