Journées de recherche sur la mémoire répartie partagée, MPR'96, Bordeaux, pp. 143-150, Mai 1996.

Mécanisme de va-et-vient réparti en mémoire dans un environnement réseau haut débit

P. Cadinot, N. Dorta, B. Folliot, P. Sens

Laboratoire MASI, Institut Blaise Pascal,
Université Pierre et Marie Curie.
4 place jussieu, 75252 Paris cedex 05, FRANCE

email: {cadinot, dorta}@ecoledoc.ibp.fr, {Bertil.Folliot, Pierre.Sens}@masi.ibp.fr

Résumé.

L'évolution des réseaux à haut débit offre de nouvelles possibilités de conception de systèmes d'exploitation répartis tirant profit de la vitesse de transfert de mémoire centrale à mémoire centrale. Cet article expose un de ces aspects, le mécanisme de va-et-vient (swap) qui vise à diminuer le temps global d'accès à la mémoire virtuelle, en déchargeant les pages non plus sur un disque local mais dans la mémoire des machines inutilisées du réseau. Ce va-et-vient réparti est destiné à des applications parallèles dédiées, s'exécutant sur un ensemble de stations de travail interconnectées par un réseau à haut débit.

Abstract.

High speed networks evolution offers new possibilities in designing distributed operating system, taking into account the increasing speed of memory to memory access. This paper addresses one of these topics, a distributed remote swap. It aims to reduce the global access time to virtual memory by swapping out pages, not on a local disk, but on the remote memory of idle machines. This remote swap is designed for dedicated parallel applications running on top of a set of workstations connected by a high speed network.

1. Motivations

L'émergence des réseaux à haut débit (Ethernet 100 Base T, ATM, FDDI) modifie la hiérarchie des temps d'accès par rapport aux réseaux classiques (Ethernet 10 Base T, Token Ring) :

Mémoire locale, disque, réseau È Mémoire locale, réseau, disque.

Ces réseaux offrent une grande bande passante et une faible latence. Dès lors, rechercher une information dans une mémoire distante devient moins coûteux qu'une lecture sur un disque local [11]. Cette caractéristique offre de nouvelles possibilités de conception de systèmes d'exploitation répartis tirant profit de la vitesse de transfert de mémoire à mémoire, telles que les caches disques répartis ou la mémoire virtuelle répartie.

Notre objectif est de fournir un support d'exécution pour des applications nécessitant de grandes quantités de mémoire. Les applications considérées sont des outils parallèles dédiés de CAO-VLSI. Ces outils accèdent à de très nombreuses structures de données (par exemple, la vérification à plat de règles de CAO pour le processeur STACS de 875.000

transistors nécessite 800 Mo de mémoire). Les performances sont donc très liées aux temps d'accès à la mémoire. Très exigeantes en mémoire, ces applications ne peuvent obtenir un espace mémoire local suffisant pour s'exécuter. La mise à disposition de l'ensemble des mémoires des stations d'un réseau permet non seulement d'améliorer les performances du mécanisme de va-et-vient (*swap*) mais également d'exécuter de nouvelles gammes d'applications. Ce mécanisme de va-et-vient réparti est en cours de développement au laboratoire MASI, et à terme, sera utilisé par la machine parallèle Multi-PC.

La section 2 présente un état de l'art des mécanismes de va-et-vient. La section 3 discute des limites et performances du va-et-vient réparti. La section 4 décrit les critères de conception de notre projet. Finalement, la section 5 conclue et donne un aperçu des résultats attendus.

2. Etat de l'Art

Cette section décrit les différentes possibilités de va-et-vient, local, distant sur disque et distant en mémoire.

Va-et-vient local dans Unix.

Si nous prenons exemple sur le système UNIX, le mécanisme de va-et-vient se décompose en deux sous problèmes : les mécanismes et les politiques [7, 8].

Les mécanismes de va-et-vient utilisent le système de mémoire virtuelle des processeurs. La mémoire virtuelle est décrite par des tables de pages. Placées en mémoire, ces tables contiennent des informations de localisation des pages (adresse locale, secteur disque), ainsi que leur état (présente, utilisée, modifiée). Évincer une page revient à la recopier sur disque puis à invalider l'entrée correspondante dans la table des pages. Les emplacements sur le périphérique de va-et-vient sont alloués à la demande. Ils ne sont libérés que lorsque le processus se termine, une page déjà présente dans l'espace de va-et-vient n'a donc pas besoin d'être réécrite si elle n'est pas modifiée. Le périphérique de va-et-vient peut être un fichier ou une partition physique d'un disque, c'est généralement cette dernière solution qui est choisie pour éviter l'indirection que représente le système de fichier.

Les politiques et stratégies de va-et-vient déterminent la granularité, le moment et les pages à évincer, de même que l'emplacement où les stocker. Les principales politiques sont le "*swap in / swap out*" et la pagination. Le "*swap in / swap out*" a pour objet d'évincer la totalité du processus de la mémoire, lorsque celui-ci est inactif depuis plus de 20s, ou lorsque le mécanisme de pagination n'arrive pas à maintenir un nombre suffisant de pages libres. Ce mécanisme a pour rôle d'évincer les pages inutilisées des autres processus. Pour cela, les tables de pages sont scrutées régulièrement à la recherche de pages à évincer. Cette scrutation dure jusqu'à l'obtention d'un quota suffisant de pages libres.

Va-et-vient sur disque distant.

Initialement, la pagination à distance a été utilisée pour des machines ne disposant pas de disque local (*diskless*). Toute page est envoyée par le réseau vers un serveur de pages [9]. Le temps de va-et-vient est alors égal au temps « accès réseau + accès disque distant ». Un tel serveur est facilement réalisable via NFS [10]. Le va-et-vient sur disque distant améliore les performances d'un va-et-vient local dans le cas où le serveur distant est équipé de disques plus rapides que les disques locaux (par exemple : architecture RAID). Cependant, ces disques peuvent difficilement être mis à la disposition de nombreux clients sans que le serveur devienne un goulot d'étranglement. En outre, la diminution du temps de pagination due aux disques plus rapides peut être absorbée par le temps de latence du réseau. Certaines méthodes de pagination procèdent par anticipation (*lookahead*), supprimant alors le temps de latence pour les pages anticipées.

Va-et-vient en mémoire distante.

Le va-et-vient se fait uniquement dans la mémoire de serveurs distants. Ce mécanisme supprime le temps de latence induit par l'accès à un disque distant et augmente la disponibilité des serveurs auprès des clients. Cela suppose, une politique de sélection de machines “serveur de mémoire” et surtout, que la somme des mémoires des serveurs soit suffisante. Dans le cas contraire, il est possible de voir la mémoire distante comme un premier niveau de stockage avant le va-et-vient sur disque. Cette méthode est souvent utilisée dans le cas de serveurs de fichiers distribués qui utilisent leur mémoire comme cache pour les disques. Les performances sont alors dépendantes du taux de présence dans le cache (*hit*), qui est de moins de 10% [12]. Ce faible taux est dû au fait que ces systèmes utilisent un même cache pour les fichiers et la pagination. Les politiques de remplacement de pages désignent les pages les moins utilisées. Les pages de la mémoire virtuelle sont donc évincées en priorité pour laisser place à celle du système de fichier.

Machines dédiées / non dédiées.

La sélection d'une machine de stockage de pages peut s'effectuer soit statiquement au préalable d'une exécution (machines dédiées), soit dynamiquement en fonction de la charge des machines et de l'évolution de l'exécution (machines non dédiées).

Dans le cas d'un va-et-vient réparti sur disque, le fort temps de latence suggère un fonctionnement statique, en particulier, il serait trop coûteux de changer de disque de stockage en cours d'exécution.

Dans le cas d'un va-et-vient réparti en mémoire les modes de fonctionnement machines dédiées ou non dédiées sont possibles. Avec des serveurs non dédiés, toutes les machines sont susceptibles de prendre le rôle de serveur de pages. Le principal problème consiste alors à collecter une connaissance globale sur l'état du système (principalement la mémoire disponible) de manière à connaître les serveurs potentiels, tout en prenant en compte les besoins de chaque processus et le caractère privé des stations de travail.

3. Limites et performances

G.Bernard et S.Hamma [2] simulent une pagination mémoire à distance. L'idée de la pagination à distance est d'utiliser la mémoire oisive des autres machines. Ils examinent différents critères (taille des processus, taux de défaut de pages, taux d'utilisation du cpu, espace mémoire alloué au va-et-vient distant) caractérisant les politiques de va-et-vient, afin de calculer l'impact que la pagination distante a sur les processus locaux et vice-versa. Ils observent qu'un processus local utilisant intensivement la mémoire et déjà sujet au va-et-vient (sur disque), n'est pas trop pénalisé (pénalité de 7%) par la pagination d'un processus distant dans son espace mémoire. L'effet sur un processus utilisant intensivement le processeur est d'autant plus important que le processeur est réquisitionné par le système de pagination : pour 40 défauts de pages/seconde (dfp/s) en lecture ou 17 dfp/s en écriture la pénalité est de 10%. Elle croit rapidement avec le nombre de fautes de pages en écriture (jusqu'à 100% pour 100 dfp/s). En revanche, l'effet des processus locaux, utilisant intensivement la mémoire, sur la pagination à distance est plus important. En effet dès 30 dfp/s on perd le bénéfice de la pagination à distance, de plus le nombre de fautes de pages que le serveur est capable de traiter diminue très rapidement. Au contraire, le taux d'occupation du processeur par les processus locaux a une influence négligeable sur le temps de réponse des processus clients. Cette simulation (réalisée sur un ensemble de stations de travail SUN, sous SunOS.4.1.2, interconnectées par un réseau Ethernet à 10Mbit/s), indique qu'il est possible de concilier les besoins des utilisateurs locaux et ceux du va-et-vient réparti.

A. P. Markatos, G. Dramitinos et K. Papachritos [6] réalisent un mécanisme de pagination à distance, en utilisant la mémoire oisive des autres stations du réseau. Toutes les stations sont des DEC-Alpha 3000, connectées par un réseau Ethernet 10 Base T. Le disque local pour la pagination est un DEC RZ55, ayant un débit de 10 Mbits/s et un temps d'accès de 16 ms. Les tests ont montré une amélioration significative : entre 39% et 112% d'accélération du temps d'exécution. Le va-et-vient représentait alors 62% du temps d'exécution. Le temps moyen de pagination (chargement ou déchargement d'une page) est de 9,05 ms en mémoire distante et de 17 ms sur le disque local. Ces 9,05 ms se décomposent en 7,2 ms de temps de transfert sur le réseau et 1,85 ms de temps de latence système. Le principal facteur limitatif étant le temps d'accès réseau, plus le réseau sera rapide et plus le temps de pagination sera réduit.

Le second facteur limitatif concerne le temps de latence du système. Ce temps peut être réduit au moyen d'un mécanisme d'écriture directe en mémoire distante (*remote write*) qui évite l'interruption des processeurs et la recopie dans les tampons du système. Ce type de protocole transfert les données directement depuis l'espace utilisateur local dans l'espace utilisateur distant. Un transfert se décompose en trois phases : (i) l'utilisateur place dans la file des requêtes les adresses locales et distantes des données à transférer; (ii) le matériel vient lire directement ces données en mémoire; (iii) la machine cible reçoit ces données et les écrit directement à l'adresse destinataire. L'émetteur est responsable de la présence des données émises, celles-ci ne doivent pas être modifiées. Pour cela, il verrouille les pages avant d'en demander le transfert. Il est aussi responsable de la validité de l'adresse destinataire. Le site destinataire doit au préalable lui envoyer ces adresses.

4. Conception du va-et-vient réparti

La section 4.1 présente le support de réalisation du mécanisme de va-et-vient réparti. La section 4.2 introduit les principes de notre solution. Les sections 4.3 et 4.4 détaillent les mécanismes, politiques et stratégies. La section 4.5 donne un aperçu des solutions possibles dans le domaine de la tolérance aux fautes.

4.1. Support de réalisation

Notre plate-forme de test initiale est composée de 5 *Pentium* 150 Mhz interconnectés par un concentrateur Ethernet 100 Base T (100 Mbits/s). Le système retenu est une version publique, modifiable et documentée du système UNIX : le système FreeBSD. Ce système dispose d'un mécanisme de pagination en mode utilisateur, ce qui permet de limiter les modifications à apporter au noyau. Cette plate-forme constitue un modèle de ce que l'on peut trouver dans de nombreux réseaux d'entreprises. Nous y avons ajouté un émulateur du protocole d'écriture distante (nommé DDSBP : *Direct Deposit Sender Based Protocol*) [4].

Par la suite, nous nous placerons dans le cadre du projet Multi-PC du laboratoire MASI, qui vise à la conception d'une machine multi-ordinateurs. Elle est composée d'un grand nombre de processeurs standards (Pentium) reliés par un réseau haut débit. Ce réseau, le *High Speed Link* (HSL) [3], utilise le routeur *Rcube* développé au laboratoire MASI. Il a pour caractéristique un très grand débit (640 Moctets/s) et un temps de latence faible (< 10 µs). Il est associé à un protocole matériel de d'écriture direct en mémoire distance nommé *Direct Deposit Sender Based Protocol* (DDSBP). Ce réseau assure des transferts de messages en quelques dizaines de micro-secondes, à comparer au 10 milli-secondes d'un réseau Ethernet 10 Base T (10 Mbits/s) standard.

Le réseau HSL de la machine MPC est un réseau de type multi-étage. Toutes les machines ne sont donc pas à égale distance les unes par rapport aux autres, nous devons donc envisager la possibilité de redistribuer les pages. Le temps libre des serveurs et du réseau peut être employé à établir des statistiques d'utilisation des pages et une éventuelle utilisation de ces résultats pour leur chargement anticipé et leur bonne localisation.

4.2. Principes

La mise en oeuvre de ce va-et-vient réparti repose sur le modèle client / serveur. Les clients, souhaitant éviter de décharger des pages sur leur disque local, utilisent la mémoire centrale de serveurs de pages. Ces serveurs sont constitués des machines peu chargées du réseau. Nous prévoyons trois étapes de conception : (1) une unique machine cliente, le reste des machines étant des serveurs dédiés, (2) plusieurs machines clientes coopérantes pour exécuter plusieurs applications, (3) un mode symétrique où les machines peuvent devenir cliente ou serveur, ou les deux à la fois. Nous ne considérons pas le cas du partage de pages entre plusieurs processus, car il suppose un modèle de mémoire virtuelle partagée répartie, en dehors de la problématique du va-et-vient réparti.

(1) La première étape consiste en une configuration statique composée d'une seule station cliente. Cette configuration a pour avantage de bien isoler les facteurs qui influencent la performance. Sa mise en oeuvre repose sur des mécanismes simples. On répartit l'espace mémoire virtuel du client sur les différents serveurs. Un simple entrelacement des pages permet de distribuer la charge des serveurs, ceux-ci sont alors désignés par un modulo sur l'adresse virtuelle (figure 1).

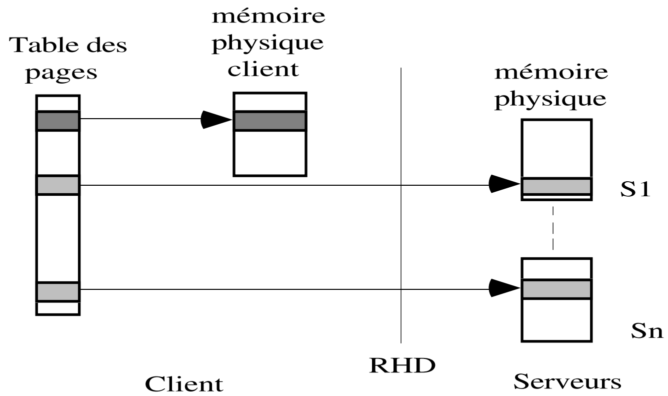

Figure 1 : Va-et-vient réparti statique à un client.

(2) Notre objectif est d'étendre ce travail à un modèle où plusieurs clients d'une même application s’exécutent sur un nombre restreint de stations (stations clientes). Les processus se trouvant sur ces stations nécessitent toujours plus de mémoire qu'il n'y en a physiquement. Ces clients utiliseront donc la mémoire des machines inutilisées du réseau (stations serveurs). Ce partitionnement est fait dynamiquement pendant l’exécution. Initialement, toutes les stations sont considérées comme serveurs. Par la suite, un processus candidat au va-et-vient réparti se fait connaître du serveur de pages local, celui-ci passe alors en mode client. Si ce serveur avait des pages en sa possession il les redistribue entre tous les autres serveurs. Contrairement au modèle précédent, il est possible que la mémoire globale ne soit pas suffisante. Nous allons donc utiliser un deuxième niveau de va-et-vient en utilisant les disques locaux des clients ou ceux des serveurs de pages afin d'y placer les pages non récemment utilisées. L'espace de va-et-vient réparti peut être vu comme une hiérarchie de mémoires (figure 2).

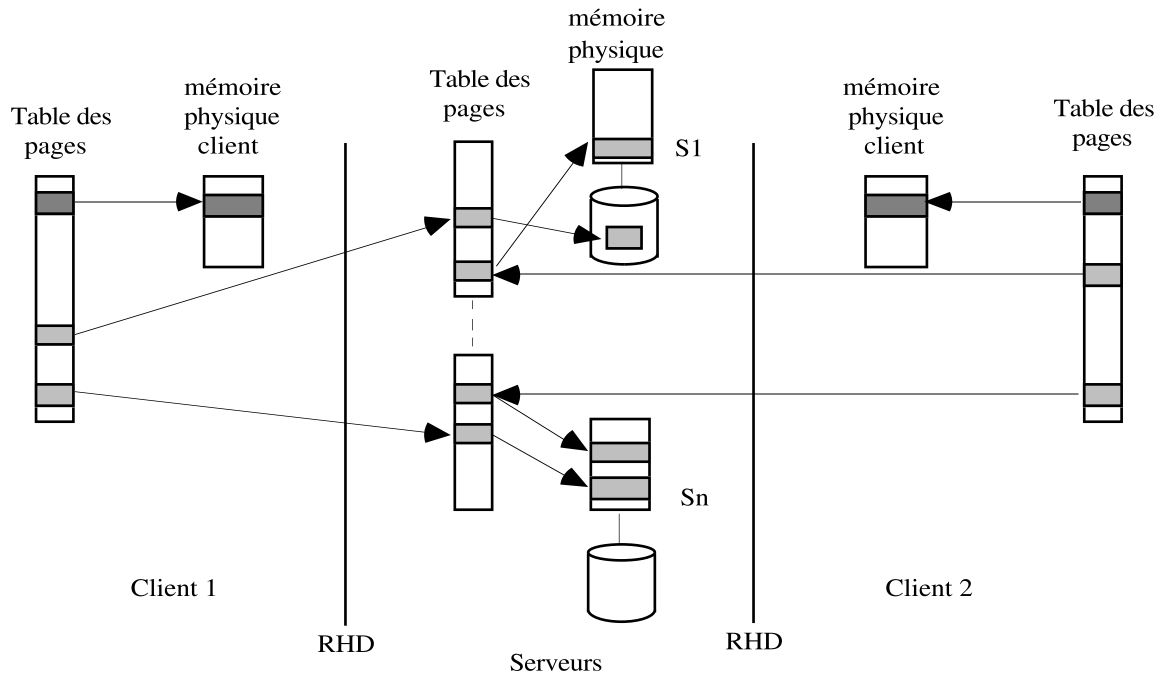

Figure 2 :Va-et-vient dynamique à deux clients.

(3) Dans le cas particulier de la machine MPC, toutes les machines sont équivalentes. Il est donc nécessaire de transformer les configurations précédentes en un modèle symétrique, où toutes les machines sont à la fois serveurs et clientes. Cela suppose un contrôle réparti dynamique du nombre de pages libres sur les serveurs.

Dans la suite de cet article nous décrirons plus particulièrement le deuxième modèle.

4.3. Mécanismes

Différencier les processus est nécessaire afin de discerner ceux qui pourront participer au mécanisme de va-et-vient réparti. La solution choisie consiste à modifier la structure des processus (champs vers le périphérique destinataire) permettant ainsi de sélectionner entre le va-et-vient standard (réservé aux processus systèmes) et le va-et-vient réparti pour chaque processus en particulier.

La conception d'un va-et-vient réparti utilise les mêmes mécanismes de mémoire virtuelle que pour un va-et-vient local. Toutefois, il faut ajouter aux structures locales de contrôle des pages des informations sur leur localisation, à savoir, un bit de localisation (local/distant), et l'identité du serveur qui gère la page. Les serveurs de pages doivent pouvoir envoyer les pages aux processus clients mais aussi être capables de déplacer (migration) des pages dynamiquement. Pour cela, ils doivent connaître pour chaque page l'identité du processus (n° de processus + adresse de la machine) propriétaire.

Le mécanisme de va-et-vient réparti modifie le va-et-vient de FreeBSD pour utiliser le protocole DDSBP (section 4.1). Un client voulant écrire une page utilise l'adresse physique fournie par le serveur pour sauver la page à distance. Symétriquement, un défaut de page provoque l'envoi d'une requête au serveur qui la détient. Cette requête contient la référence de la page, ainsi que l'adresse d'un emplacement physique où placer localement la page manquante. Le transfert de plus d'une page (anticipation) est effectué en donnant au serveur la liste des pages et des emplacements physiques.

4.4. Politiques & Stratégies

Pour réagir rapidement aux demandes de déchargement, les clients conservent un quota de pages (un*pool* contenant une liste d'adresses physiques) fourni par les machines identifiées comme serveurs. Après le transfert d'un grain de pages sur le serveur, celui-ci peut renouveler le *pool* du client en lui fournissant un nouveau quota. Les stratégies de va-et-vient doivent considérer la granularité des transferts, ainsi que le moment et les pages à évincer. Dans le va-et-vient réparti il faut en plus sélectionner une machine de destination.

Grain ? Il faut trouver un compromis entre l'optimisation de la bande passante du réseau (gros grain) et sa disponibilité (grain fin). Regrouper les pages lors des échanges entre clients et serveurs permet de diminuer le temps de latence des protocoles systèmes et réseaux proportionnellement au nombre de pages. De plus, le protocole DDSBP permet de transmettre plusieurs pages en générant une seule interruption sur le destinataire. En outre, le taux de localité (*working-set*) de l'application est un facteur essentiel à la définition de la taille du grain optimale. Un bon taux de localité permet d'augmenter la taille du grain. En revanche, l'envoi d'une requête suite à un défaut de page peut alors se trouver bloqué par l'envoi préventif d'un tel bloc de page.

Quand ? Afin de maintenir un ensemble de pages libres en mémoire et aussi de réduire le temps de latence au chargement, il est nécessaire de faire de l'évincement préventif. Un seuil calculé comme un pourcentage de la mémoire définit la limite au-delà de laquelle il faut activer ce mécanisme d'évincement préventif. Un seuil trop bas implique un gaspillage de la mémoire et à terme des va-et-vient inutiles; au contraire, un seuil trop haut risque de laisser le système bloqué en attente de pages disponibles. Par exemple, ce seuil représente 6,25% de la mémoire utilisateur sur Système V Release 4 [8] et de 12,5% dans 4.3 BSD [7].

Qui ? Les algorithmes de choix d'une page à évincer ne changent pas pour un va-et-vient réparti. Les algorithmes généraux tel que : FIFO, NRU permettent de sélectionner la page, mais on évince (en écrivant les pages modifiées) tout le bloc dans lequel se trouve la page désignée (grain). L'utilisation du mécanisme de va-et-vient est restreint à l'ensemble des processus désignés explicitement. Enfin, nous avons choisi de ne jamais utiliser le mécanisme de va-et-vient réparti pour les processus systèmes (voir 4.5).

Où ? Contrairement au va-et-vient local, la localisation d'un serveur est cruciale. Elle est résolue en partie par le mécanisme d'identification qui délimite l'ensemble des serveurs. Par la suite, les systèmes clients choisissent de façon circulaire le serveur sur lequel placer leurs pages. Chaque serveur décide après la fin d'un transfert de réallouer ou non un quota de pages en fonction de sa place mémoire disponible.

4.5. Tolérance aux fautes

La tolérance aux fautes dans notre système de va-et-vient réparti peut-être assurée à quatre niveaux. Dans le premier niveau, l'utilisation du va-et-vient réparti par les processus critiques est interdite. Cela permet aux autres systèmes de rester intègres en présence de la panne d'un serveur.

Le deuxième niveau consiste à identifier les processus affectés par une panne et à les abandonner. Le va-et-vient réparti continue ainsi à fonctionner, et les applications peuvent être redémarrées.

Le troisième niveau est l'intégration d'un gestionnaire de tolérance aux fautes (comme GatoStar [13]), qui permet de continuer l'exécution en présence de fautes au moyen d'un mécanisme de point de reprise où toutes les pages sont sauvegardées sur support stable.

Finalement, le quatrième niveau consiste à intégrer la tolérance aux fautes directement dans le gestionnaire de va-et-vient réparti au moyen d'un mécanisme de réplication des pages, ou par calcul de parité. Cette solution est la plus efficace mais est la plus complexe à mettre en oeuvre, c'est à terme celle que nous comptons implanter.

5. Conclusion

Nous avons décrit les principes et les mécanismes d'un va-et-vient réparti en mémoire, dédié à des applications ayant des besoins en mémoire dépassant les capacités d'une seule machine. Nous avons détaillé les problèmes et les solutions retenues pour notre va-et-vient réparti dans le cadre de machines dédiées, liées par un réseau à haut débit avec mécanisme d'écriture distante (*remote write*). L'interface logicielle pour simuler l'écriture distante sur un réseau Ethernet 100 Base T a été développée.

Nous avons proposé trois étapes de conception. Une configuration statique avec 1 client et N serveurs de pages, qui permettra d'isoler les facteurs prépondérants sur les performances. Une configuration dynamique avec X clients coopérants pour l'exécution d'une seule application, qui permettra d'étudier le modèle dynamique du va-et-vient réparti, ainsi que la gestion du stockage à deux niveaux (mémoire distante, disque local). Finalement, une configuration symétrique adaptée à la machine parallèle MultiPC permettra à chaque processeur de jouer indifféremment le rôle de client et/ou de serveur.

Compte tenu des bons résultats des travaux déjà effectués [6], nous espérons obtenir des gains d'environ 10 pour un réseau local à haut débit à comparer au va-et-vient standard, et d'environ 100 dans le cadre de la machine MPC. Par la suite, nous allons étudier l'extension de ce mécanisme à d'autres types d'applications, ainsi que la mesure de l'impact du va-et-vient réparti sur les performances globales du système.

6. Références

[1] A. L. Pinto Ferreira de Sousa et J. O. Roque Nascimento Pereira.
USWAP/RSWAP - Implementação de um *swapper* remoto em modo utilizador.
Universidade do Minho - Portugal, Juillet 1994.

[2] G. Bernard et S. Hamma.
Remote memory paging in network of workstations.
In Proc. Symposium UNIX Users Group, International Conference on Open Systems : Solutions for Open World, à Moscou - Russie, Avril 1994.

[3] J. Lecler, F. Potter, J.L. Desbarbieux, F. Wajsbürt et A. Greiner.
PCI-DDC / Spécifications (rapport).
Laboratoire M.A.S.I., Avril 1996.

[4] F. Potter, P. David, A. Fenyö.
Interface Logicielle pour la machine M.P.C.
Laboratoire M.A.S.I. / PRiSM, 1996.

[5] T. E. Anderson, D. E. Culler, D. A. Patterson, et l'équipe NOW .
A Case for NOW (Networks of Workstations).
IEEE Micro, February 1995, pp 54-64.

[6] A. P. Markatos, G. Dramitinos, K. Papachritos.
Implementation and evaluation of a Remote Memory Pager.
Institute of Computer Science (ICS), Foundation for Research & Technology - Hellas (FORTH), Technical Report FORTH/ICS 129, Grèce, Mars 1995.

[7] S. J. Leffler, M. K. McKusik, M. J. Karels, J. S. Quarterman.
The Design and Implementation of the 4.3BSD UNIX Operating System.
Addison-Wesley, 1990.

[8] B. Goodheart, J. Cox.
The magic garden explained. The internals of UNIX System V Release 4. An open system design.
Prentice Hall, 1994.

[9] R. Minnich. Mether-NFS.
A Modified NFS Which Support Virtual Shared Memory.
In Proc. 4th Symposium on Experiences on Distributed and Multiprocessor Systems (SEDMS-IV),
San Diego, CA., Septembre 1993.

[10] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli et R. Y. Wang.
Serverless Network File Systems.
In ACM press, Fifteenth ACM Symposium on Operating Systems Principles, Volume 29, Numéro 5, Decembre 1995, pp 109-126.

[11] C.A Thekkath.
System Support for Efficient Network Communication.
Departement of Computer Science and Engineering, University of Washington, Seattle, rapport technique 94-07-02, Juillet 1994.

[12] M. G. Backer, J.H.Hartman, M. D. Kupfer, K.W. Shirriff et J. K. Ousterhout.
Measurements of a Distributed File System.
In ACM press, Thirteenth ACM Symposium on Operating Systems Principles, Volume 25, Numéro 5, Octobre 1991, pp 198-212.

[13] B. Folliot, P. Sens, P.G. Raverdy.
Plate-forme de répartition de charge et de tolérance aux fautes pour applications parallèles en environnement réparti.
Calculateurs Parallèles, Editions Hèrmes, volume 7 n°4, 1995, pp 345-366.