THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut National Polytechnique de Toulouse (INP Toulouse)*

Présentée et soutenue le *07/10/2013* **par :**

AEIMAN GADAFI

Gestion mémoire dans une infrastructure répartie

JURY

Noël DE PALMA	Professeur d'Université	Président du Jury
Jean-François MÉHAUT	Professeur d'Université	Rapporteur
Pierre SENS	Professeur d'Université	Rapporteur
Laurent BROTO	Maître de Conférences	Co-Directeur de thèse
Daniel HAGIMONT	Professeur d'Université	Directeur de thèse

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (UMR 5505)
Directeur(s) de Thèse :
Daniel HAGIMONT et *Laurent BROTO*
Rapporteurs :
Jean-François MÉHAUT et *Pierre SENS*

A celle que je porterai toujours dans mon cœur :
à ma famille

Votre temps est limité, ne le gâchez
pas en menant une existence qui n'est
pas la vôtre.
***Steve Jobs**. Extrait d'un discours à*
Stanford en 2005.

Je tiens tout d'abord à remercier les membres du jury qui ont accepté d'évaluer mon travail. Je remercie mes deux rapporteurs : monsieur Méhaut et monsieur Sens pour le temps qu'ils ont consacré à rapporter cette thèse, mais également pour l'échange très intéressant qu'on a eu pendant ma soutenance. Je remercie monsieur Noel Depalma d'avoir accepté de présider le jury et d'examiner cette thèse. Je remercie monsieur Laurent Broto, Co encadrant de ce travail de thèse, pour l'aide technique qu'il m'a portée au cours de ma thèse et pour tous les efforts et le temps qu'il a consacré pour améliorer mon manuscrit.

Je ne trouverai jamais les mots pour exprimer ma gratitude envers monsieur Daniel Hagimont, mon directeur de thèse. Ses conseils et ses encouragements ont permis à ce travail d'aboutir. Ses capacités scientifiques et ses compétences étaient mon grand support. Merci infiniment à toi *dan* pour ta patience et la grande confiance que tu m'as accordées. Merci pour toutes les discussions, les idées et les remarques (toujours piétinantes et constructives) qui font avancer.

Au-delà du jury, je tiens à remercier tous ceux qui ont permis à ces travaux d'aboutir, par leurs conseils, leurs contributions et leurs encouragements.

Je remercie les admirables secrétaires d'IRIT/ENSEEIHT (les deux Sylvies) pour les services rendus. Je remercie l'ensemble de l'équipe SEPIA, et en particulier l'unité ENSEEIHT (Alain, Brice, Sun, Suzy, Larissa) pour l'ambiance de travail très agréable et pour tous les moments qu'on a partagés ensemble.

Je tiens à remercier profondément l'ensemble des personnels du service des affaires culturelles à l'ambassade libyenne à Paris pour les services rendus pendant mon séjour en France.

Je tiens bien évidemment à remercier toute ma famille pour son soutien et encouragement durant ces années d'étude. Je remercie en particulier mon père, ma femme et mes enfants de m'avoir toujours soutenu, compris et encouragé, et m'avoir remotivé dans les moments difficiles de la thèse. Très grand merci à toi *Sana* d'avoir été toujours à côté de moi, d'avoir supporté mon absence et d'avoir pris beaucoup de responsabilités à ma place.

Je ne peux pas oublier de remercier tous mes amis qui m'ont accompagnée tout au long de mon parcours : mes « potes » de toujours (Eihab, Ibrahim, Ali, Khalid et Hussein) et mes amis en France (notamment Anwar, Agila, Imhemmed, Yaser, Salah, les Hassans, Othman et Moktar).

De nos jours, de plus en plus d'organisations déploient des infrastructures matérielles telles que des clusters ou des grilles. Elles sont utilisées pour héberger des services internet communs tels que l'email, les réseaux sociaux ou le commerce électronique ou pour exécuter des applications scientifiques telles que les simulations nucléaires ou les prédictions météorologiques. La capacité de traitement et de stockage demandée pour répondre à la charge de travail de ces applications ne peut être fournie que par le biais de ces infrastructures matérielles.

Ces infrastructures matérielles embarquent des systèmes d'exploitation, qui peuvent potentiellement coopérer dans le but de gérer au mieux les ressources disponibles. Ces systèmes gèrent alors l'allocation des ressources aux applications en fonction des besoins de ces dernières. Ces systèmes visent à garantir la qualité de service et en même temps à gérer de façon optimale les ressources dans le but de limiter les coûts, notamment l'énergie.

La communauté scientifique s'est intéressée à la problématique de la gestion des ressources. De nombreuses approches ont été proposées et des solutions ont été mises en œuvre. En réalisant un état de l'art de ces approches, nous constatons que la plupart d'entre elles s'intéressent à la gestion des nœuds dans l'objectif de répartir les calculs d'une façon adéquate pour exploiter de manière optimale la charge processeur. La gestion globale de la mémoire dans de telles infrastructures n'a pas été suffisamment étudiée. En effet, la mémoire est souvent considérée comme une ressource avec une capacité théoriquement illimitée grâce aux mécanismes de swap, mais ces derniers ont des conséquences importantes sur les performances des applications et le coût de fonctionnement de l'infrastructure.

Dans cette thèse, nous étudions la conception et l'implantation d'un service de gestion globale de la mémoire dans une infrastructure matérielle. Ce service de gestion de mémoire doit éviter le gaspillage de mémoire et ne doit pas pénaliser les performances des applications hébergées. Nous proposons un service de swap à distance permettant à une machine, plutôt que swapper sur son disque local, de swapper sur la mémoire distante d'une autre machine ayant de la mémoire disponible. Les pages distantes peuvent être déplacées dynamiquement afin d'équilibrer la charge entre les machines. Ceci permet de mutualiser la mémoire et d'économiser les ressources. Un prototype a été implémenté et évalué.

Mots-clés : gestion ; mémoire ; swap ; infrastructure ; répartie ; collaboration ; ressources.

Nowadays, more and more organizations are deploying large scale infrastructures such as clusters or grids. They are used to host common Internet services such as email, social networks, e-commerce applications or to run scientific applications such as nuclear simulations and weather predictions. Processing power and storage capacities satisfying the workload of these applications can only be provided by such infrastructure.

The operating systems deployed on these nodes manage the allocation of application resources and can potentially cooperate in order to manage the available resources according of the application needs.

The scientific community is usually interested in the resource management problematic. Many approaches have been proposed and solutions have been implemented. However, we find out that most of them focus on the node management in order to adequately distribute calculations to optimally exploit the CPU load. The global memory management in such infrastructures has not been enough studied. Indeed, memory is often considered as a resource with a theoretically unlimited capacity thanks to the swap capabilities, but swapping has a significant impact on the system performance and the operation cost.

In this thesis, we study the design and the implementation of a global memory service management in a large scale infrastructure. This memory management service must avoid wasting memory resources and should not penalize the performance of hosted applications. It is based on remote swap mechanisms. A prototype has been implemented and evaluated.

Keywords : Memory management ; swap ; cluster ; resources management ; collaboration.

Table des matières

Chapitre 1

Introduction

Les infrastructures réparties, telles que les clusters ou les clouds, sont utilisées de plus en plus fréquemment par beaucoup d'organisations. Ces infrastructures, composées d'un grand nombre de machines, permettent en agrégeant les ressources de ces machines de fournir une capacité élevée de calcul et de stockage. Les entreprises déploient de telles infrastructures pour se confronter à la grande charge de travail imposée sur les services qu'elles proposent tel que le commerce électronique, les réseaux sociaux, etc... Les centres de recherche utilisent également ces infrastructures pour fournir la capacité requise pour exécuter des applications scientifiques demandant une grande capacité de calcul telles que les simulations nucléaires, les recherches de pétrole, les prédictions météorologiques, etc...

Un de principaux défis pour les fournisseurs d'infrastructures est la gestion de ressources. Il s'agit de l'opération qui permet d'affecter les ressources aux applications en fonction de leurs besoins. Puisque les ressources sont limitées, elles doivent être partagées pour répondre aux besoins des applications et les satisfaire en terme de capacité de calcul, de communication, de mémoire, etc... La gestion de ressources est une opération essentielle pour les fournisseurs puisqu'elle contrôle la qualité de services fournis par l'infrastructure (par exemple le temps de traitement maximal d'une requête) et contrôle également les coûts de fonctionnement de l'infrastructure (par exemple la consommation d'énergie).

De nombreuses approches ont été proposées et des solutions ont été mises en œuvre pour implanter des systèmes efficaces de gestion de ressources. Elles s'intéressent à garantir la qualité de services et à optimiser les ressources dans le but de limiter les coûts de fonctionnement de l'infrastructure. La majorité de ces solutions se base sur l'utilisation et la gestion des machines dans l'objectif de répartir les calculs d'une façon adéquate pour exploiter de manière optimale la charge processeur.

Dans le cadre de cette thèse, nous nous intéressons à la gestion de la mémoire vive dans une infrastructure matérielle. En effet, la gestion de la mémoire dans une infrastructure matérielle est souvent traitée au niveau local (chaque machine

traitant ce problème indépendamment) en étendant la mémoire de la machine par le biais du disque local. Ceci peut être géré soit à travers les techniques de mémoire virtuelle et de swap implantées au niveau du système d'exploitation, soit au niveau de l'application avec des opérations explicites sur disque. Grâce à ces techniques, la mémoire devient une ressource théoriquement illimitée.

Cependant, cette gestion locale a trois inconvénients :

- les disques ont des capacités de stockage illimitées, mais iles sont trop lents par rapport à la mémoire vive ce qui influence directement les performances des applications ;
- puisqu'il n'y a pas de coopération entre les machines, on n'exploite pas la mémoire éventuellement disponible dans l'infrastructure (sur les autres machines) ;
- les infrastructures de nos jours utilisent de plus en plus des machines qui n'ont pas de disques « e.g. mode *diskless* » et se servent de serveurs de stockage « SAN » accessibles via le réseau. Dans ce cas, les coûts d'utilisation des disques sont plus importants, car il s'agit d'accès à des disques distants.

Il existe également des approches qui s'intéressent à la gestion de la mémoire au niveau global (niveau de l'infrastructure) en s'appuyant sur des techniques de migration de processus ou de répartition de charge, mais elles présentent des limites notamment par rapport à l'intrusivité, à la performance et à l'utilisation optimale des ressources.

Dans cette thèse, nous étudions la conception et l'implantation d'un service de gestion globale de la mémoire dans une infrastructure matérielle. Nous proposons plus précisément un service de swap à distance permettant à une machine, plutôt que d'utiliser son disque local en tant que support de swap, de swapper sur la mémoire distante d'une autre machine ayant de la mémoire disponible. Ce service permet également un équilibrage dynamique de la charge du swap entre les machines distantes hébergeant le swap. Il permet d'éviter le gaspillage de la ressource mémoire et améliore les performances des applications tout en optimisant l'utilisation des ressources dans l'infrastructure.

Plan de document

Ce rapport de thèse est organisé de la façon suivante :

- **Chapitre 2** .
  Ce chapitre est divisé en trois parties. Dans un premier temps, nous présentons les infrastructures matérielles, leurs utilisations et les principaux types d'infrastructures matérielles qui existent aujourd'hui. Ensuite, nous présentons les systèmes de gestion de ressources avec leurs objectifs et leur fonctionnement.

Enfin, nous analysons les limites des gestions de la mémoire dans les infrastructures matérielles afin d'introduire notre problématique.

– **Chapitre 3**
Le chapitre 3 présente un état de l'art des travaux de recherche qui ont été effectués autour de la gestion mémoire dans une infrastructure matérielle. Nous commençons par présenter nos critères d'évaluation. Puis, nous présentons les différentes classes de travaux dans le domaine en les illustrant par les implantations majeures pour chacune d'elle. Nous terminons par une synthèse qui récapitule ces approches et leurs limites par rapport à nos critères.

– **Chapitre 4**
Nous présentons dans ce chapitre notre contribution qui a pour objectif de fournir un système de gestion globale de la mémoire dans une infrastructure matérielle en adressant les limites présentées précédemment. Il est divisé en trois parties : nous commençons par présenter les principes généraux de l'approche et les choix de conception que nous avons faits. Puis, nous présentons l'utilisation de notre approche : sa configuration et son installation. Enfin, nous décrivons l'implantation de ce système (appelé ASM).

– **Chapitre 5**
Le chapitre 5 présente les résultats des évaluions effectuées avec ASM. Nous présentons trois séries d'évaluation : les mesures concernant les mécanismes de base d'ASM, une évaluation avec une configuration statique et une autre avec une configuration dynamique.

– **Chapitre 6**
Nous concluons ce manuscrit et donnons la liste des évolutions que nous envisageons en vue de rendre notre approche plus efficace.

Première partie

Contexte d'étude

Chapitre 2

Contexte et problématique

Contents

2.1 Infrastructure matérielle haute performance

2.1.1 Introduction

Une infrastructure matérielle haute performance [1] est un ensemble de machines, appelé nœuds [2], reliées par un réseau. Ils sont conçus pour atteindre des hautes performances, en particulier en termes de vitesse de calcul, dans l'objectif d'améliorer le temps d'exécution et la disponibilité des applications qui nécessitent de telles capacités [6].

Le concept d'infrastructure matérielle est apparu pour résoudre des problèmes nécessitant une capacité importante de calcul. Pour fournir de telles capacités, on s'appuyait auparavant sur des supercalculateurs extrêmement coûteux et complexes, qui utilisaient des processeurs et des logiciels spécifiques. Grâce aux infrastructures matérielles, on peut atteindre à faible coût en agrégeant les capacités de plusieurs nœuds, composés des processeurs et des systèmes d'exploitation standards, les performances des supercalculateurs.

Les infrastructures matérielles sont devenues aujourd'hui populaires et sollicitées par beaucoup d'entreprises et centres de recherche. Selon une étude publiée par le cabinet américain *Gartner* [3], le marché mondial d'infrastructure matérielle a généré un chiffre d'affaires de 58,6 milliards de dollars en 2009 et devrait doubler entre 2010 et 2014. En 2014 *Gartner* estime qu'il devrait approcher les 150 milliards de dollars. Ceci est dû principalement à l'émergence de services internet tels que l'e-mail, les moteurs de recherche, les réseaux sociaux et le commerce électronique, mais également aux applications scientifiques telles que les simulations nucléaires, les recherches de pétrole, les prédictions météorologiques, etc...

2.1.2 Utilisation

La raison initiale pour le développement et l'utilisation des infrastructures matérielles était de fournir une puissance élevée de calcul [60]. La liste *TOP500* [4] qui classe les 500 super-ordinateurs les plus rapides chaque année comprend souvent de nombreuses infrastructures matérielles, c'est le cas par exemple de l'infrastructure matérielle *K-computer* qui a été considérée comme la machine la plus rapide du monde en 2011 [64].

1. Pour simplifier, nous utilisons le terme (infrastructure matérielle)
2. Dans la suite de ce document, nous utilisions plutôt ce terme qui désigne une machine physique permettant d'exécuter des processus.
3. www.gartner.com
4. www.top500.org

Les infrastructures matérielles peuvent être utilisées pour garantir un accès constant aux services et fournir une haute disponibilité. Ils permettent de répartir la charge de travail [5] parmi un grand nombre de serveurs pour assurer les services en évitant au maximum les indisponibilités. Ceci permet de fournir des systèmes informatiques tolérants aux pannes [6]. Lorsqu'un nœud tombe en panne, ceci ne fait que réduire la puissance globale de l'infrastructure matérielle.

2.1.3 Classification

Aujourd'hui, il existe plusieurs types d'infrastructures matérielles. Les orientations et les préoccupations diffèrent selon le type d'infrastructure. Chaque type a des caractéristiques propres à lui. On trouve donc différentes approches d'implantations.

Les infrastructures matérielles peuvent être classées selon différents paramètres. Ils concernent les nœuds, le réseau, et l'utilisation de l'infrastructure. Nous détaillons ces paramètres comme suit :

- **Homogénéité/hétérogénéité :**
  au sein d'une infrastructure matérielle, les nœuds peuvent être homogènes ou hétérogènes. Ceci peut concerner les caractéristiques matériels des nœuds « type de processeur, taille mémoire, etc. . . » ou les aspects logiciels « système d'exploitation installé sur les nœuds, système de gestion de fichiers, etc. . . » ;
- **Vitesse et fiabilité du réseau :**
  la manière dont les nœuds sont interconnectés joue un rôle important dans la classification du type d'infrastructure matérielle. Les nœuds peuvent être déployés dans la même zone géographique par un réseau local ou sur une grande zone géographique par un réseau étendu. Ceci a une influence sur le type d'infrastructure matérielle car les conditions de fonctionnement ne sont pas les mêmes « à savoir la vitesse, le taux de transfert de données, le protocole, etc. »

Nous présentons dans les sections suivantes les grands types d'infrastructures matérielles qui existent aujourd'hui. Nous présentons les propriétés de chaque type ainsi des exemples d'utilisations.

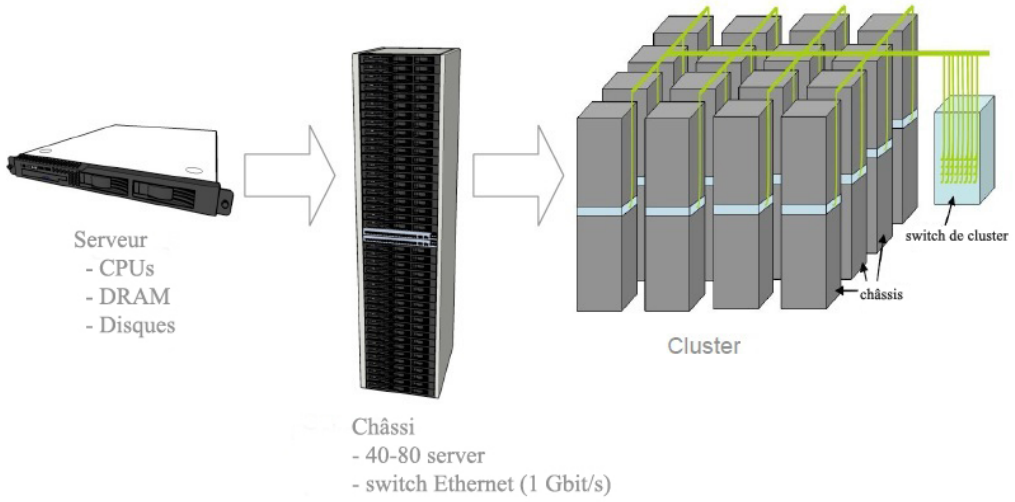

FIGURE 2.1 – Exemple d'un cluster

2.1.3.1 Grappe de serveurs

La grappe de serveurs (le cluster) est un ensemble de nœuds homogènes, déployés physiquement au même endroit, et inter-connectés par un réseau local rapide et fiable. Ils peuvent être considérés comme un seul système.

La figure 2.1 montre un exemple de l'architecture interne d'un cluster proposé par Google [23]. Nous pouvons voir qu'il est composé d'un ensemble des Serveurs Lame [7]. Ils sont mutualisés dans des châssis [8] capable d'accueillir de 40 à 80 serveurs et de les relier par un réseau Ethernet de 1 Gbit/s. Le cluster est composé de plusieurs châssis reliés par un réseau filaires à fibre optique à 10 Gbit/s.

Nous pouvons classer les clusters selon la rapidité d'interconnexion du réseau et la puissance des nœuds. Nous en citons deux types : les clusters haute performance, tels que les clusters *Cray* [9] ou les clusters *Fujitsu* [10], et les clusters à faible coût, tels que les clusters *Beowulf* [60].

Un cluster haute performance, qu'on appelle supercalculateur, est un ensemble d'ordinateurs à haute performance, reliés par un réseau ultra rapide. Ce type de cluster est conçu pour atteindre les performances les plus importantes, en particu-

5. La charge de travail est la quantité de traitement que le nœud doit effectuer à un moment donné.
6. La tolérance aux pannes désigne une méthode de conception permettant à un système de continuer à fonctionner, au lieu de tomber complètement en panne, lorsque l'un de ses composants ne fonctionne plus correctement.
7. en anglais *Serveur Blade* : des nœuds conçus pour un très faible encombrement.
8. en anglais : *Rack*
9. www.cray.com
10. www.fujitsu.com

lier en termes de vitesse de calcul. Ils sont utilisés pour toutes les applications qui nécessitent une très forte puissance de calcul comme les prévisions météorologiques.

Un cluster à faible coût, tel que *Beowulf*, est un cluster homogène dont les composants sont des PC standards inter-connecté par un réseau rapide. Le système a été au départ développé par la *NASA* [11]. Il est maintenant couramment utilisé dans beaucoup d'organisations car ce type de clusters peut être construit à un faible coût par rapport aux clusters haute performance.

2.1.3.2 Grille

Le terme grille[11] a été introduit aux États-Unis durant les années 1990 pour décrire une infrastructure de calcul distribuée, utilisée dans les projets de recherche scientifique et industriels [27, 39]. Elle mutualise un ensemble de nœuds géographiquement distribués sur plusieurs sites pour atteindre un objectif commun. Chaque site peut être un ensemble de clusters. Chacun est administré indépendamment des autres. Ces sites collaborent ensemble pour résoudre des problèmes complexes [27].

Les nœuds de grille peuvent être hétérogènes et ils proviennent de plusieurs domaines administratifs différents. Le taux de transfert de données est généralement inférieur à celui-ci des clusters.

La figure 2.2 montre un exemple d'une grille pour l'infrastructure *Grid5000* [17]. Sur cette figure nous pouvons constater que *Grid5000* est composé de plusieurs cluster « en forme de nuage », distribués sur de différentes villes de France.

Les grilles peuvent être classées en deux catégories : les grilles qui sont composées d'un ensemble de clusters, tel que *Grid5000*, et les grilles P2P, tel que *BOINC*[3] et *SETI@home*[4] où l'idée est de lier de nombreux ordinateurs de bureau pour fournir une puissance de traitement globale. Dans le premier cas, les nœuds sont homogènes (dans un cluster) et la communication est fiable, tandis que dans le deuxième, les nœuds sont hétérogènes et de faible puissance, ils sont liés par un réseau à faible vitesse et non fiable (souvent via l'Internet).

2.1.3.3 Nuage

Il existe plusieurs définitions de nuage informatique (cloud), nous prenons la définition de *CISCO* [7] «*Le Cloud est une plateforme de mutualisation informatique fournissant aux entreprises des* ***services à la demande avec l'illusion d'une infinité de ressources.*** »

11. En anglais : *Grid*

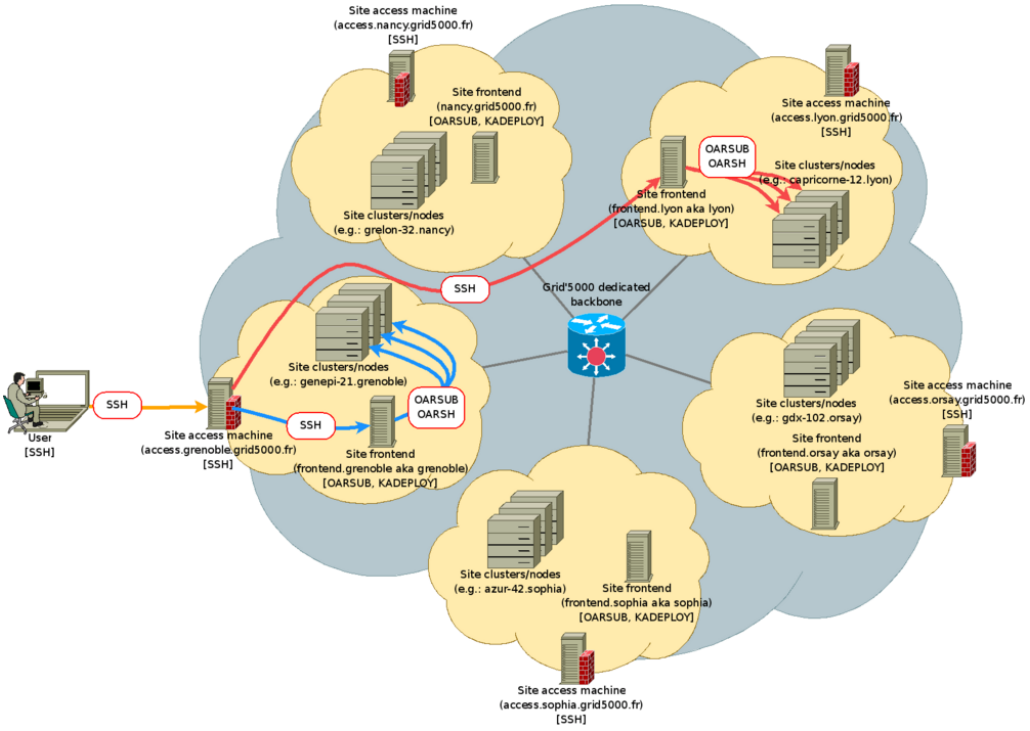

FIGURE 2.2 – Exemple d'une grille (*Grid5000*)

Cette définition met l'accent sur les deux propriétés essentielles des clouds. Tout d'abord, les clouds sont basés sur la distinction entre clients et fournisseurs. Puis, la définition met l'accent sur une propriété propre aux clouds : ils donnent la possibilité de contrôler les services et les ressources allouées aux clients. Selon les besoins des clients, les ressources peuvent varier dynamiquement pour répondre à une charge de travail variable.

Les clouds se basent principalement sur les techniques de virtualisation [10], c'est-à-dire la possibilité d'exécuter plusieurs systèmes d'exploitation sur une machine physique, chacun de ces systèmes d'exploitation s'exécutant sur une machine virtuelle (VM de l'anglais : Virtual Machine) et n'ayant pas conscience de partager le matériel. Ces techniques offrent des avantages intéressants pour les fournisseurs de clouds, notamment l'isolation et la migration de VM entre les nœuds physiques sans interrompre les applications en cours d'exécution [34].

Avant les clouds, les clients s'adressaient aux *Datacenters* ou ils pouvaient louer des machines physiques. Cette opération était lourde et pouvait prendre du temps. Les clouds ont simplifié cette opération. Ainsi, louer une machine ne prend que quelques secondes. De plus, elle peut être rendue dès qu'on en a plus besoin de manière aussi simple que la location. Ce système de location rapide a été généralisé. Ainsi, il existe aujourd'hui trois modèles de base qui peuvent être utilisés sur un cloud selon les ressources allouées aux clients. La figure 2.3 montre ces trois modèles

ainsi que le modèle traditionnel de *Datacenter* et les ressources fournies aux clients dans chaque modèle. On peut ainsi distinguer :

- **IaaS (*Infrastructure as a service*)** : le fournisseur met à disposition uniquement des ressources matérielles (machine, réseaux, disque) et le client gère le système et les applications ;
- **PaaS (*Platform as a service*)** : le client ne gère et ne contrôle ici que ses applications métier, le reste est délégué au fournisseur de services ;
- **SaaS (*Software as a service*)** : c'est l'ultime niveau de transfert vers le fournisseur, le client ne gère plus que ses données métier, il utilise les applications fournies par le fournisseur. Il s'agit d'utiliser le logiciel à la demande.

Les deux premiers modèles (IaaS et PaaS) sont généralement utilisés dans le cadre d'une relation entre entreprises (*Business To Business (B2B)*). Le modèle SaaS est au contraire utilisé généralement dans des relations entre une entreprise et le grand public. (*Business To Consumer (B2C)*).

Il existe plusieurs mises en œuvre du cloud telles qu'*Amazon EC2*[12] (Iaas), *Google Apps*[13] (SaaS) ou *Google App Engine*[14] (PaaS).

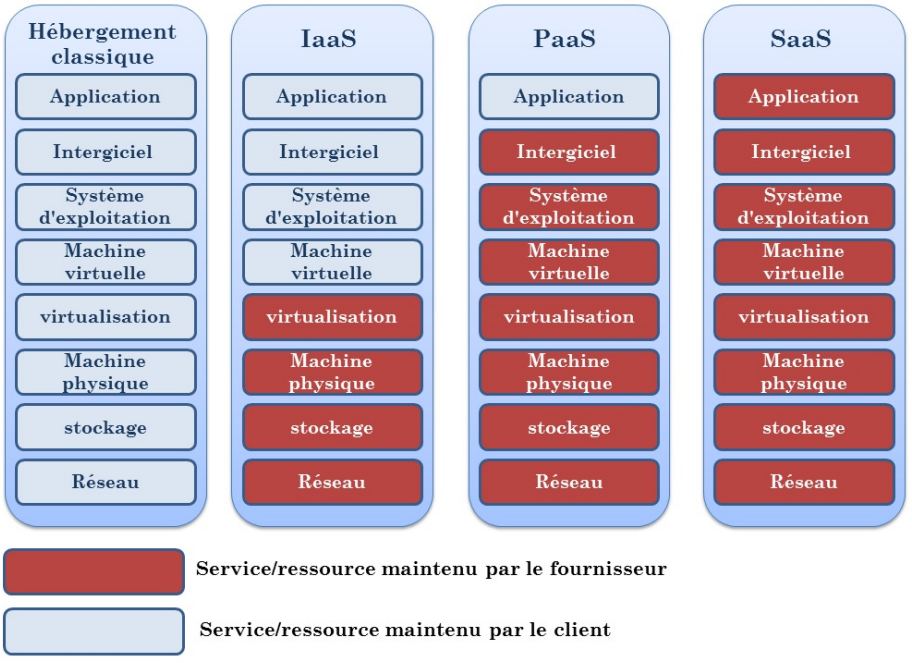

FIGURE 2.3 – Modèles de cloud

12. aws.amazon.com/ec2
13. www.google.com/intx/fr/enterprise/apps/business/
14. developers.google.com/appengine

2.1.4 Synthèse

Nous avons présenté le concept d'une infrastructure matérielle. Nous avons pu voir les raisons pour lesquelles ces infrastructures sont largement utilisées. Nous avons présenté les grands types d'infrastructures qui existent aujourd'hui, les propriétés de chaque type ainsi des exemples d'utilisation.

Quelque soit le type d'infrastructure matérielle, le nombre de ressources [15] est fini. Les ressources doivent donc être partagées parmi les utilisateurs de l'infrastructure. Il faut donc un système de gestion de ressources qui permet d'allouer ces ressources pour satisfaire les besoins d'utilisateurs.

Nous présentons dans la section suivante les objectifs et les contraintes posées lors de la gestion des ressources. Nous expliquons ensuite le fonctionnement d'un système de gestion de ressources. Nous terminerons par la présentation des stratégies de gestion de ressources ainsi des approches d'implantation.

15. Une ressource peut être définie comme les moyens dont dispose un nœud pour exécuter des processus. Les principaux types de ressources sont le processeur, ou le *Central Processing Unit* (CPU), la mémoire vive, ou la *Random Access Aemory* (RAM), et les entré/sorties.

2.2 Gestion de ressources

La gestion de ressources dans une infrastructure matérielle est l'opération qui permet d'affecter les ressources aux applications. C'est une préoccupation pour les fournisseurs de tout type d'infrastructure puisque les ressources sont limitées. Elles doivent donc être partagées pour répondre aux besoins des applications et les satisfaire en termes de capacité de calcul, de communication, de mémoire, etc...

Pour ce faire, il faut avoir un système spécifique permettant de garantir les accès adaptés à des ressources partagées. Ce système est appelé **le système de gestion de ressources (SGR)**. Nous décrivons par la suite les objectifs de tels systèmes ainsi leurs contraintes. Nous présentons leurs fonctionnements puis nous décrivons des stratégies qui ont été proposées pour les implanter.

2.2.1 Contraintes et objectifs

Avant de présenter les objectifs qu'un SGR doit atteindre, il nous semble important de définir trois concepts qui sont utilisés dans la suite de ce document : la qualité de service, l'accord de niveau de service et les objectifs de niveau de services.

- ***Qualité de service (Quality of Services QoS)***
  La qualité de service désigne la capacité d'un service à répondre par ses caractéristiques aux différents besoins de ses utilisateurs ou consommateurs[16].
  Dans une Infrastructure matérielle, la QoS a pour but de garantir de bonnes performances aux applications en termes de disponibilité, débit, délais de transmission, taux de perte, etc...

- ***Accord de niveau de service (Service Level Agreement SLA)***
  Un SLA est la formalisation d'un accord négocié entre deux parties. C'est un contrat qui définit la QoS requise entre un prestataire et un client. Il précise de manière claire les objectifs de performance et de qualité. Le fournisseur du service s'engage ainsi par contrat sur une disponibilité de l'outil vis-à-vis des utilisateurs.

- ***Objectifs de niveau de services (Service Level Objectives SLOs)***
  Les SLOs définissent des caractéristiques des services qui seront fournis à un client en termes qualitatifs. Ils sont considérés comme des moyens permettant de mesurer la QoS dans l'objectif d'éviter le malentendu entre les deux parties. Ils peuvent être composés d'une ou plusieurs mesures de QoS. Par exemple, un SLO de disponibilité peut dépendre de plusieurs composants, chacun d'entre eux pourra avoir une mesure de QoS de disponibilité propre.

16. Définition d'*AFNOR* (association française de normalisation) site : www.afnor.org

Le SGR a un certain nombre de contraintes qu'il doit satisfaire. Ces contraintes influencent la manière dont le système alloue des ressources. Elles peuvent être distinguées en deux catégories : les contraintes fonctionnelles et les contraintes non fonctionnelles.

Les contraintes fonctionnelles représentent la QoS que l'infrastructure doit fournir pour satisfaire les utilisateurs tel qu'elle est spécifiée dans le SLA. Ces contraintes concernent la performance fournie par l'infrastructure en termes de capacité de traitement, disponibilité, etc... Ceci peut être le temps de traitement maximal d'une requête (temps de latence) ou le nombre de requêtes traitées par seconde (le débit). Un premier objectif du SGR consiste alors à allouer les ressources requises pour assurer le respect ces spécifications SLA.

Le SGR doit prendre en compte un autre type de contraintes exigées par le fournisseur de l'infrastructure. Il s'agit de contraintes non fonctionnelles. Cela peut concerner par exemple la sécurité de l'infrastructure ou la consommation d'énergie. Les fournisseurs définissent ce type de contraintes principalement pour réduire le coût de fonctionnement, notamment par rapport à la consommation d'énergie qui devient un enjeu majeur pour la plupart des fournisseurs. Selon l'agence américaine de protection de l'environnement [24], on estime que le secteur des infrastructures matérielles a consommé environ 61 milliards kWh en 2006 pour un coût d'électricité total de 4,5 milliards de dollars. En outre, la consommation d'énergie de ces infrastructures est estimée avoir doublé entre 2000 et 2006 et le développement des centres d'hébergement vont amplifier cette augmentation. Les fournisseurs et la communauté scientifique ont pris conscience de ces contraintes et s'y intéressent de plus en plus.

La gestion efficace des ressources en termes de performances et d'énergie doit prendre en charge ces deux types de contraintes généralement contradictoires : d'un côté il faut rendre les services aux utilisateurs en améliorant leurs qualités et en respectant les indications de QoS spécifiées dans les SLA mais de l'autre côté, il faut réduire les coûts de fonctionnement de l'infrastructure en utilisant le minimum de ressources.

2.2.2 Rôle et fonctionnement

La fonction principale d'un SGR consiste en l'allocation et l'affectation des ressources où et quand elles sont nécessaires le plus efficacement possible. Pour ce faire, le SGR doit surveiller en permanence les ressources et les activités qui les utilisent.

La figure 2.4 montre l'architecture globale d'un SGR. Dans cet exemple, le système agit sur un ensemble de ressources « R1,R2 et R3 » qui sont sollicitées par des services ou des applications « S1, et S2 ». Nous trouvons dans cette figure une

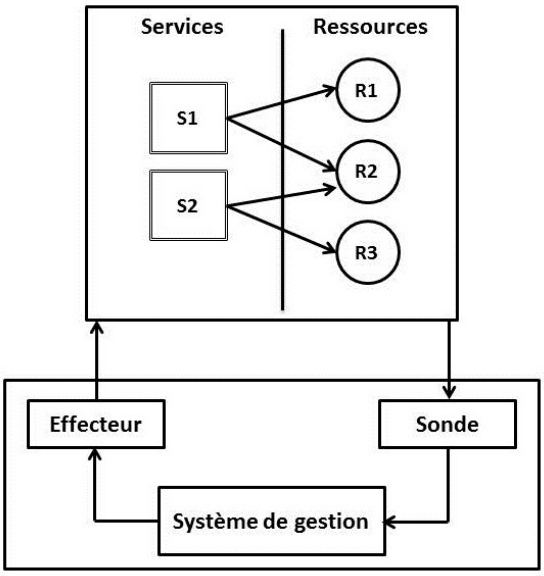

FIGURE 2.4 – architecture globale d'un SGR

boucle classique du système réactif qui est composé de trois étapes : mesure, décision, action.

Le SGR a besoin d'information pour pouvoir gérer les ressources. Pour cela, il utilise ce que nous appelons des sondes. Une sonde fournit au SGR les informations nécessaires à son fonctionnement. Ces informations peuvent concerner l'état d'utilisation des ressources comme par exemple la charge CPU ou mémoire, ou bien des informations concernant des indicateurs de QoS provenant des applications comme par exemple le temps de réponse d'un serveur. À partir de ces informations, le SGR peut prendre des décisions selon des conditions et des objectifs prédéfinis qui forment ce que nous appelons la politique de gestion. Ces décisions sont par la suite appliquées aux ressources et aux applications par le biais d'effecteurs.

Le niveau de l'implantation du SGR ainsi que la stratégie de gestion influencent les décisions prises par le SGR et les effecteurs disponibles pour les appliquer. Nous donnons des exemples de décisions et discuterons plus en détail les stratégies de gestion dans la section suivante.

2.2.3 Stratégies de gestion

Plusieurs approches et solutions ont été proposées pour répondre aux contraintes présentées précédemment. Le SGR peut intervenir à deux niveaux différents : au niveau du nœud et au niveau de l'infrastructure. Nous distinguons ainsi deux classes de stratégies de gestion de ressources : les stratégies locales et les stratégies globales.

Nous détaillons ces deux types dans les paragraphes suivants. Pour chaque classe de stratégies, nous présentons les principales approches.

2.2.3.1 Stratégies locales

Les stratégies locales de gestion de ressources s'appuient sur la gestion de ressources au niveau du nœud. Elles concernent la gestion de composants matériels qui appartiennent au nœud tel que le processeur, la mémoire, le disque, etc... Cette classe de stratégies est gérée au niveau du système d'exploitation de chaque nœud. Chacun de ces systèmes gère ses ressources indépendamment des autres. Pour des stratégies locales complexes qui s'adaptent aux besoins particuliers des fournisseurs, ces derniers peuvent utiliser des systèmes spécifiques de gestion de ressources (par exemple [20]).

La gestion locale de ressources consiste historiquement à répondre aux contraintes fonctionnelles en contrôlant les demandes d'utilisation des ressources provenant des applications. Le SGR accorde les ressources aux applications en contrôlant ces demandes éventuellement contradictoires. Il décide quand et quelle quantité de ressources est mise à disposition des applications et comment ces ressources sont utilisées.

Aujourd'hui, les systèmes d'exploitations s'adaptent pour intégrer d'autres contraintes non fonctionnelle comme l'énergie. Pour ce faire, ils peuvent reposer sur des technologies offertes par des ressources à travers à l'interface *Advanced Configuration and Power Interface*(ACPI) [17] qui est une interface permettant d'envoyer des signaux à ces différents composants matériels pour contrôler leur fonctionnement. Plusieurs composants offrent ces technologies, par exemple : les processeurs proposent la technique *dynamic voltage and frequency scaling* (DVFS) [49] qui permet de contrôler sa performance et consommation d'énergie selon les conditions de charge. La mémoire et les cartes de réseau proposent des techniques similaires [51, 30].

Nous donnons quelques exemples de gestion locale de ressources dans le paragraphe suivant.

Approches d'utilisation

Le SGR alloue des ressources aux applications en cours d'exécution. Si une application est chargée et demande plus de ressources, le SGR peut, dans la mesure du possible, récupérer des ressources qui ont été allouées à d'autres applications et qui ne sont pas utilisées. Un exemple de cette approche est les techniques de mémoire virtuelle. En effet, le SGR peut allouer plus de mémoire lorsqu'elle est chargée en évacuant une partie des données non utilisées vers le disque. Un autre exemple de cette approche est dans le cas où il y a plusieurs machines virtuelles s'exécutant sur

17. www.acpi.info

une machine physique. Le SGR définit des quotas pour chaque VM en termes de ressources. Lorsqu'une VM consomme ses ressources et en demande plus, le SGR peut décider de lui fournir des ressources disponibles d'une autre VM. Ceci peut être fait au niveau du processeur [33], mais aussi au niveau de la mémoire [63].

Pour la gestion de contraintes non fonctionnelles comme l'énergie, le SGR met les ressources sous-utilisées dans un état qui consomme moins d'énergie au prix de performances moindres. Si ces ressources sont ensuite sollicitées par des processus, le SGR adapte leurs états en changeant leurs modes de fonctionnement pour augmenter leurs capacités. Par exemple, en augmentant la fréquence du processeur, le SGR donne plus de capacité de calcul à une application.

2.2.3.2 Stratégies globales

Les infrastructures matérielles d'aujourd'hui sont généralement composés de nœuds hétérogènes avec des capacités variables de traitement et de stockage. Ces nœuds hébergent des applications différentes. Cela implique que la charge de travail ainsi que l'utilisation des ressources varient d'un nœud à l'autre au sein de l'infrastructure.

Les stratégies globales de gestion de ressources se basent sur la charge de travail soumise sur l'ensemble de l'infrastructure. Elles s'appuient sur un ensemble de techniques destinées à optimiser l'utilisation des ressources de chaque nœud de l'infrastructure en répartissant la charge de travail sur les différents nœuds afin de mieux utiliser les ressources disponibles et valoriser la performance.

Le SGR dans ce cas peut être un système d'exploitation distribué (par exemple MOSIX [9]) ou d'un logiciel spécifique de gestion de ressources (par exemple TUNe [14], Jade [13] ou Diet [18]). Ce système de gestion agit sur les ressources matérielles de l'infrastructure à différents niveaux : composant, nœud, un ensemble de nœuds, ou bien sur les applications déployées sur les nœuds.

Le fonctionnement de ces approches est en général décomposé en trois étapes illustrées sur la figure 2.5. Il s'agit des mêmes principes que nous avons présentés précédemment mais au niveau de l'infrastructure. Les nœuds sont observés à travers d'outils de surveillance qui permettent de vérifier la charge de travail soumise sur l'ensemble des nœuds. Le système de gestion prend ensuite des décisions de gestion de ressources se basant sur les SLA. Enfin il met en œuvre ces décisions sur l'infrastructure.

Pour prendre en charge les contraintes de consommation d'énergie, le SGR doit minimiser le taux d'utilisation des nœuds quand cela est possible. Une approche pertinente, appelée la consolidation de serveurs, consiste à orienter la charge de travail vers le plus petit ensemble de nœuds qui permet de répondre aux SLA. Les nœuds qui ne participent pas au traitement sont mis dans un état qui consomme moins d'énergie : « hors tension, suspension, etc. . . »

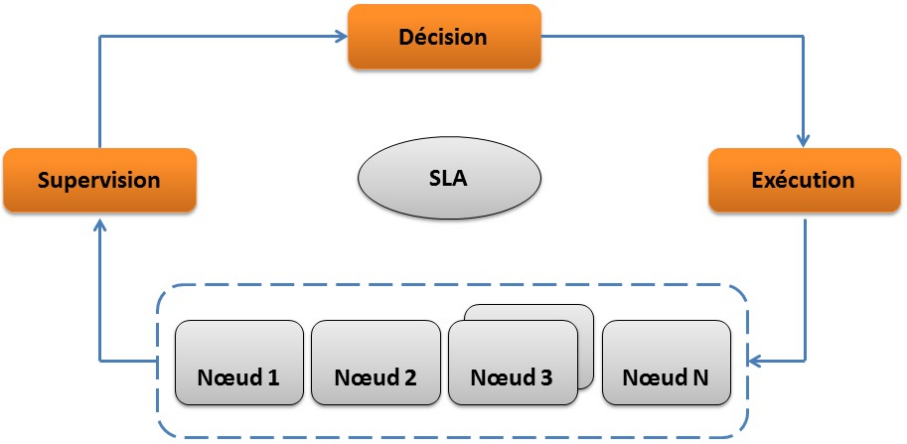

FIGURE 2.5 – approches générale de gestion de ressources

Nous présentons dans les paragraphes suivants deux approches utilisées pour la gestion globale des ressources dans une infrastructure matérielle.

Migration d'activité

Cette approche agit sur les activités exécutées dans l'infrastructure. Le principe est de déplacer les activités dans l'infrastructure pour respecter les SLA. Si le SGR constate qu'une activité sur un nœud demande plus de ressources qu'il n'y en a de disponibles sur le nœud, il déplace cette activité sur un autre nœud possédant plus de ressources disponibles et qui peut assurer les besoins de l'activité en question. Avant d'effectuer le déplacement, le SGR doit trouver un nœud avec des ressources disponibles pour pouvoir héberger l'activité. Cette approche nécessite une connaissance globale de l'état de l'infrastructure pour pouvoir prendre des décisions cohérentes et éviter les déplacements inutiles.

Dans la cadre de la consolidation, cette approche est utilisée pour concentrer la charge de travail sur le plus petit ensemble de nœuds qui peut respecter les SLA. Les nœuds non utilisés sont mis hors tension. Le SGR a recours à ces nœuds lorsqu'il n'y a pas de nœuds actifs pouvant assurer les besoins des activités.

La migration d'activité est une solution très courante aujourd'hui grâce aux techniques de virtualisation, notamment la migration à chaud de VM qui offre avec un coût minimal une solution intéressante pour déplacer les VM dans l'infrastructure. Les VM permettent de migrer des activités qui ne sont pas conçues pour être migrées. En effet la migration concerne le conteneur de l'activité qui est la VM et l'activité n'est pas au courant de cette opération. Plusieurs solutions ont été implantées pour mettre en œuvre cette approche, en prenant notamment en compte la charge du processeur (par exemple [47, 61, 59]).

La réplication

La réplication est une approche utilisée afin de fournir des systèmes à haute performance [53]. Il s'agit de répliquer une activité sur un ou plusieurs nœuds dans l'objectif de fragmenter le traitement et augmenter la performance.

En utilisant cette approche, lorsque le SGR constate qu'il y a une activité en manque de ressources, il la réplique sur un autre nœud. Ainsi, la charge de travail est divisée entre plusieurs nœuds. Cette réplication nécessite une connaissance des applications exécutées sur les nœuds, et demande souvent le changement de celles-ci pour intégrer le nouveau répliqua. Les applications doivent donc être prévues pour être répliquées.

Un exemple de cette approche est la plate-forme Java 2 Enterprise Edition (JEE) qui définit un modèle pour le développement d'applications Web [57]. JEE est une architecture à plusieurs niveaux composée d'un tiers serveur web, un tiers serveur d'application et un tiers serveur base de données. Pour pouvoir répliquer les serveurs, un logiciel particulier, appelé distributeur de charge (*load balancer*) est installé devant chaque ensemble de serveurs répliqués. Il équilibre dynamiquement la charge entre les réplicas.

Comme l'approche précédente, celle-ci peut être utilisée dans la cadre de la consolidation (par exemple [28, 31, 29]).

2.2.4 Synthèse

Nous avons présenté les systèmes de gestion de ressources ainsi que les objectifs et les contraintes de tels systèmes.

La problématique de gestion de ressources est centrale pour les fournisseurs d'une infrastructure matérielle. Beaucoup d'approches et de solutions existent aujourd'hui pour la gestion de ressources. La majorité de ces solutions se base sur l'utilisation et la gestion des processeurs. En effet, le processeur est considéré comme l'unité principale de capacité de calcul et ainsi de performance fournie par l'infrastructure. De plus, plusieurs études montrent que le processeur est le composant qui consomme le plus d'énergie au sein d'un nœud [43]. L'optimisation de son utilisation est importante pour atteindre les objectifs de gestion de ressources.

Dans le cadre de cette thèse, nous nous intéressons à la gestion de la mémoire vive dans une infrastructure matérielle. En effet, la gestion de la mémoire dans une infrastructure matérielle est souvent traitée au niveau local à travers les techniques de mémoire virtuelle et de swap. Grâce à ces techniques, la mémoire devient une ressource théoriquement illimitée. Nous pensons qu'une stratégie globale de gestion de mémoire pourrait améliorer les performances de ces techniques locales.

Dans la section suivante, nous présentons les motivations de notre travail de recherche et de montrons les limites d'une gestion locale de la mémoire.

2.3 Problématique

L'objectif général de cette thèse est de contribuer à la construction de systèmes de gestion de ressources dans une infrastructure matérielle et en particulier à la gestion de la mémoire.

En effet, malgré l'augmentation de la capacité mémoire des machines, les besoins des applications modernes sont supérieurs aux ressources des machines. Par exemple, la croissance rapide de l'Internet a créé de vastes quantités d'informations disponibles en ligne. Cette information doit être traitée, analysée et liée. La gestion, l'accès et le traitement de cette grande quantité de données exigent un volume élevé d'espace mémoire et pourraient grever les ressources de mémoire [44].

Plusieurs approches ont été proposées pour une gestion efficace de la mémoire. Les fournisseurs des infrastructures s'appuient principalement sur des approches locales qui consistent à étendre la mémoire en s'appuyant sur les disques locaux. Nous présentons ces approches dans la section suivante. Nous commençons d'abord par présenter les mécanismes de mémoire virtuelle. Puis, nous présentons les approches locales de gestion de mémoire. Nous décrivons enfin les limites de ces approches lors de l'utilisation dans une infrastructure matérielle.

2.3.1 Mémoire virtuelle

Quel que soit le système d'exploitation, les techniques de base de gestion de la mémoire sont les mêmes. Ainsi, les systèmes d'exploitation associent une mémoire virtuelle propre à chaque processus. Il s'agit d'une technique de gestion de mémoire développée pour les systèmes multitâches dans les années 1960[35]. Elle virtualise les diverses formes de l'architecture de stockage de données (telles que la mémoire vive et l'espace disque) et autorise ainsi les applications à être conçues comme s'il n'y avait qu'un seul type de mémoire. De plus, cette technique donne l'illusion à l'application que toute la mémoire dont elle a besoin est directement utilisable. Ceci permet aux applications de s'exécuter dans un environnement matériel possédant moins de mémoire centrale que nécessaire.

Le principe de cette technique repose sur les mécanismes d'adressage. En effet, chaque donnée de la mémoire est accédée par son adresse. Il existe deux types d'adressages : l'adressage physique pour lequel à chaque adresse correspond un et un seul emplacement dans la mémoire physique, et l'adressage virtuel qui permet de disposer de plusieurs espaces d'adressage par un mécanisme de translations d'adresses géré au niveau de la MMU *(Memory Management Unit)*. Un exemple d'un système de mémoire virtuel est montré dans la figure 2.6. Dans cette figure, nous pouvons constater que l'espace de mémoire virtuelle est divisé en espaces uniformes, appelés pages, chacune identifiée par son numéro. La mémoire physique est divisée de

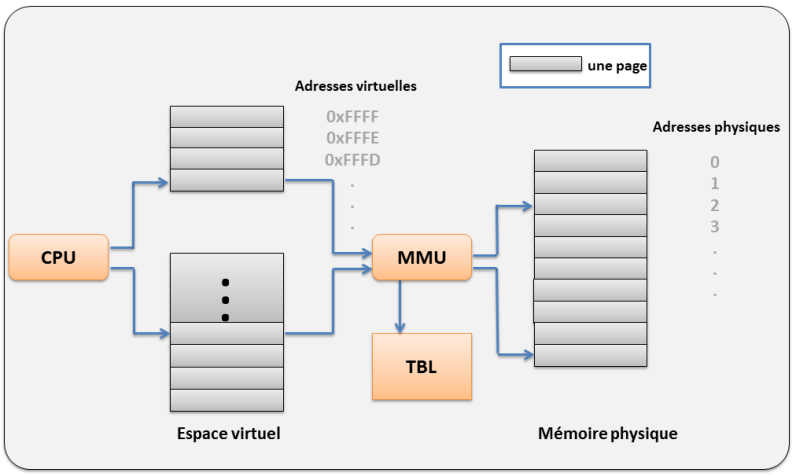

FIGURE 2.6 – Translation d'une adresse virtuelle en adresse physique

la même manière à un ensemble de cadres (en anglais : *frame*). Le gestionnaire de mémoire gère la translation d'adresses entre les pages et les cadres. Une adresse utilisée par un processus ne correspond plus directement à un emplacement dans la mémoire physique. En effet, pour obtenir l'adresse réelle d'une donnée (son adresse physique) le système utilise une table qui associe à chaque page de la mémoire virtuelle l'adresse du cadre en mémoire physique ou une erreur signifiant que la donnée n'est associée à aucune mémoire physique.

Le gestionnaire de mémoire déplace les pages entre la mémoire principale et les périphériques d'échanges (périphériques de swap), qui sont généralement des partitions de disques ou des fichiers spécifiques appelés les fichiers de swap. Lorsque la mémoire est trop chargée et qu'il y a un besoin d'espace, le gestionnaire choisit des pages victimes et les envoie vers le disque (opération appelée *swap-out*). Leur espace est ainsi libéré pour les pages les plus demandées. Les pages qui sont sur le disque seront rapatriées en mémoire (opération appelée *swap-in*) lorsqu'un processus en a besoin. Il existe plusieurs algorithmes pour choisir des pages à évacuer. Ces algorithmes sont classés en deux familles : les algorithmes dépendants de l'utilisation des données : par exemple : LRU (Least Recently Used), LFU (Least Frequently Used), et les algorithmes indépendants de l'utilisation des données : par exemple FIFO (First In First Out).

2.3.2 Approches locales de gestion de mémoire

Les approches locales de gestion de mémoire consistent à étendre la mémoire locale d'un nœud en utilisant le disque pour stocker les données. Ceci peut être fait à deux niveaux : soit au niveau du système d'exploitation à travers les techniques de mémoire virtuelle, ou au niveau de l'application. Dans les deux cas, ces approches

sont implantées par défaut et ne demande pas d'efforts de la par d'administrateurs d'infrastructure. Nous les présentons dans les paragraphes suivantes.

2.3.2.1 Swap sur disque

Pour le traitement de grandes masses de données, l'approche traditionnelle pour obtenir de la mémoire supplémentaire est de laisser le gestionnaire de mémoire du système se charger de l'échange des données entre la mémoire principale et le disque au travers de techniques de mémoire virtuelle. Lorsque le gestionnaire de mémoire n'arrive pas à maintenir un nombre suffisant de pages libres, il commence à évacuer des pages qui ne sont pas utilisées vers le swap. Ainsi, les pages seront sauvegardées sur le disque local et leur emplacement dans la mémoire sera disponible. Cette opération continue jusqu'à l'obtention d'un quota suffisant de pages libres.

Depuis son apparition dans les années 1960, cette stratégie est implantée par défaut dans la plupart des systèmes d'exploitation.

L'avantage principal de cette approche est qu'elle ne demande aucune intervention ni de la part du développeur de l'application ni de la part des administrateurs de l'infrastructure car qu'elle est gérée au niveau du système d'exploitation. C'est une stratégie valable pour tout type d'application. C'est le système d'exploitation qui charge/décharge les données de la mémoire.

2.3.2.2 Gestion anticipée de mémoire

Cette stratégie est basée sur des algorithmes spéciaux, appelés les algorithmes de mémoire externe, ou les algorithmes *out-of-core*. Ce sont des algorithmes conçus pour traiter de très grandes quantités de données qui ne peuvent pas être maintenues dans la mémoire principale d'un ordinateur [62].

Le principe de ces algorithmes consiste à optimiser la manière dont l'application cherche et accède aux données stockées sur les disques. Cette optimisation est gérée au niveau de l'application, c'est-à-dire que c'est l'application qui stocke sur disque au fur et à mesure les données qui sont générées au cours de l'exécution pour garder une quantité de mémoire disponible, et éviter donc de provoquer un swap éventuel généré par le système de mémoire virtuelle.

Les algorithmes *out-of-core* sont utilisés dans le domaine du calcul scientifique. L'utilisation des données dans ce type d'application est rigoureuse et le comportement peut être prévu à l'avance vu qu'il ne dépend pas de facteurs externes à l'application. Ainsi, les applications ont la capacité de placer efficacement les données générées au cours du calcul entre la mémoire et le disque.

La gestion anticipée de mémoire présente des avantages notables de performance par rapport à la stratégie de swap implantée au niveau système d'exploitation. Ceci a été montré par plusieurs études (par exemple [22, 58, 25]). En effet, les algorithmes de gestion de mémoire implantés au niveau du système d'exploitation sont conçus pour être génériques, mais ils ne sont pas codés avec une logique spécifique à l'application.

Bien qu'ils soient bénéfiques pour la performance, ces algorithmes sont conçus pour résoudre des problèmes liés à l'utilisation de mémoire d'une application ce qui les rend très spécifiques et difficilement réutilisables. En plus, les développeurs de ces applications écrivent une version distincte de l'application avec des appels explicites d'entrée/sortie. Ceci est une opération compliquée, car elle implique souvent une importante restructuration du code et peut avoir un impact négatif sur l'algorithme de calcul.

2.3.3 Limites des approches locales

Les approches locales de gestion de mémoire ne sont pas efficaces quand elles sont utilisées dans une infrastructure matérielle notamment en termes de performance. En effet, l'échange sur le disque peut gravement affecter la performance globale des applications. Les périphériques utilisés comme des supports du swap sont souvent des disques locaux. Ces disques ont des capacités de stockage quasi illimitées, mais ils sont trop lents par rapport à la mémoire. Selon une étude qui a été menée par *Google*®[23], le disque est environ 100 fois plus lent que la mémoire. Les applications sont donc pénalisées car elles doivent attendre longtemps avant d'avoir de la mémoire disponible.

De plus, les approches locales de gestion de mémoire pénalisent les performances des applications. En fait, l'utilisation de la mémoire dans une infrastructure matérielle peut varier considérablement d'un nœud à l'autre au cours de l'exécution. Ceci est dû à de multiples facteurs tels que les applications qui sont exécutées sur chaque nœud, les variations de la charge de travail, les phases d'exécution, etc... Il peut y avoir des nœuds qui ont de la mémoire disponible et d'autres qui n'en ont pas. Ces derniers sont obligés d'utiliser le mécanisme de swap sur disque ce qui pénalise les performances.

Par exemple, la figure 2.7 montre l'utilisation de la mémoire au cours d'une journée typique de travail dans un centre de données selon une étude menée par *INTEL*®[56]. Comme on peut le voir sur la figure, la mémoire globale du centre atteint 437TB. Toutefois, seulement 69% de cette mémoire globale est utilisée. Bien que l'utilisation globale est loin d'être à 100%, il y a environ 68TB de données résidant dans les disques en tant que support de swap. Cela démontre que certains nœuds surutilisent leurs mémoires et donc utilisent le swap, tandis que d'autres ont de l'espace mémoire disponible. Cette mémoire libre ne sera pas utilisée par les nœuds chargés vu que les gestions de la mémoire sur chaque nœud sont indépendantes.

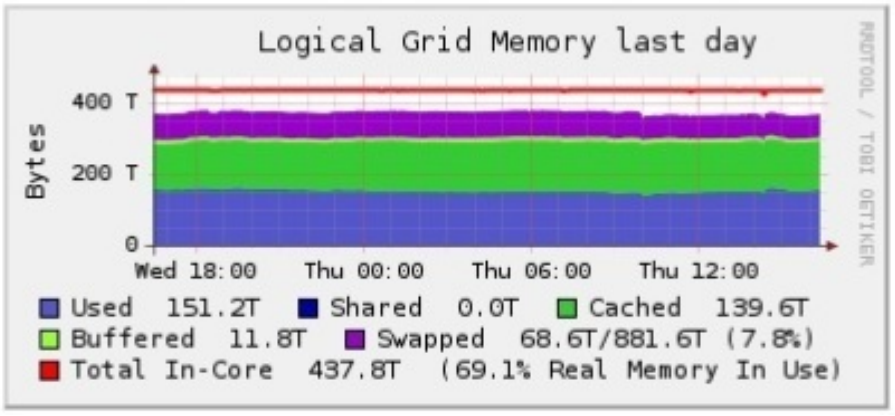

FIGURE 2.7 – Utilisation de mémoire dans un centre de données

Le manque de coopération entre les nœuds induit une perte de performance locale, voire globale lorsqu'on exécute une application répartie. Cette perte pourrait être évitée en exploitant la mémoire non utilisée au sein de l'infrastructure pour éviter les opérations sur le disque, qui sont coûteuses en ce qui concerne la performance.

Enfin, ces approches ne peuvent pas être utilisées dans beaucoup d'infrastructures matérielles actuelles qui utilisent des nœuds qui n'ont pas de disques (mode *diskless*) et se servent de réseau de stockage «*Storage Area Network* (SAN)» pour des raisons de consommation d'énergie et de performance.

2.3.4 Synthèse

Les approches locales de gestion de mémoire, qui sont basées naturellement sur l'extension de mémoire à travers des disques, ne sont pas adaptées à la gestion de mémoire dans une infrastructure matérielle. Elles pénalisent la performance globale des systèmes, gaspillent des ressources, et ne sont pas compatibles avec les infrastructures matérielles d'aujourd'hui.

Il est donc intéressant d'imaginer un système global de gestion de mémoire qui serait capable d'éviter l'utilisation de disque et de profiter de la mémoire disponible dans l'infrastructure, et qui prendrait en charge les contraintes exigées par les systèmes d'aujourd'hui, notamment en termes de performance et d'énergie.

Nous présentons dans le chapitre suivant les différentes approches de gestion de mémoire qui ont été proposées pour la gestion globale de mémoire. Ces approches montrent des limites et ne permettent pas de répondre aux exigences de gestion de ressources présentées précédemment. Nous choisissons et présentons des critères d'évaluation de ces approches. Nous terminons par un récapitulatif des différentes approches de gestion de mémoire présentées.

Chapitre 3

État de l'art

Contents

Ce chapitre présente les travaux de recherche qui ont été effectués autour de la gestion de la mémoire dans une infrastructure matérielle de type cluster. Nous présentons les différentes classes de travaux dans le domaine en les illustrant par les implantations majeures pour chacune d'elle.

Ce chapitre est divisé en trois parties : dans un premier temps, nous présentons un aperçu général des approches globales de gestion de mémoire puis nous présentons des critères sur lesquels nous nous basons pour étudier les différentes approches (section 3.1). Nous présentons ensuite les stratégies de gestion de mémoire ainsi que les approches d'implantation dans les sections 3.2 et 3.3. Nous terminons avec une synthèse qui montre une vue d'ensemble de ces approches 3.4.

3.1 Introduction

3.1.1 Plan

Dans la section 2.2.3.2, nous avons présenté les principes des stratégies globales de gestion de ressources. En ce qui concerne la mémoire, ces stratégies visent à optimiser l'utilisation de la mémoire dans l'infrastructure matérielle. Elles s'appuient sur l'état des nœuds et les ressources disponibles dans l'infrastructure pour permettre de répondre aux besoins des applications en termes de mémoire.

Dans ce chapitre, nous présentons et analysons les stratégies globales de gestion de mémoire. Dans notre présentation, nous commençons par introduire le concept général de chaque stratégie. Ensuite, nous présentons les principaux systèmes d'implantation. Nous étudions chaque système à travers les critères suivants :

- **L'efficacité :**
  l'objectif principal de la gestion de mémoire est d'améliorer la performance de l'infrastructure matérielle. La performance dans la gestion de mémoire est évaluée en fonction du temps d'accès aux informations gérées par le système de gestion de mémoire (SGM). La performance est dégradée quand le SGM cherche des informations qui ont été stockées sur des disques par le système de mémoire virtuelle. Les SGMs dans une infrastructure matérielle essaient donc d'éviter les opérations sur disque pour améliorer les performances.
  Pour chaque SGM que nous présentons, nous évaluons si la manière dont il gère la mémoire est inefficace et présente des situations ou il y a une perte non justifiée de performance. Ceci arrive lorsqu'il existe dans l'infrastructure de la mémoire principale disponible sur certains nœuds et qu'il existe d'autres nœuds qui manquent de mémoire et qui sont obligés d'utiliser les disques. Cette perte pourrait être évitée en exploitant la mémoire non utilisée au sein de l'infrastructure pour éviter les opérations sur le disque ;
- **L'utilisation optimale des nœuds :**
  Dans la section 2.2.3.2, nous avons introduit la consolidation des serveurs qui est une technique proposée pour la gestion optimale et dynamique de ressources. Cette technique répond aux contraintes non fonctionnelles posées lors de la gestion de l'infrastructure matérielle notamment la gestion d'énergie en consolidant la charge de travail sur un petit ensemble des nœuds quand cela est possible. Ceci permet par la suite d'éteindre ou de suspendre les nœuds non utilisés.
  Dans notre évaluation des systèmes de gestion de mémoire, nous regardons pour chaque système s'il prévoit l'utilisation optimale des nœuds. Nous étudions si l'algorithme de gestion de mémoire permet de concentrer la charge de travail de mémoire sur le minimum de nœuds pour éviter le gaspillage de ressources.

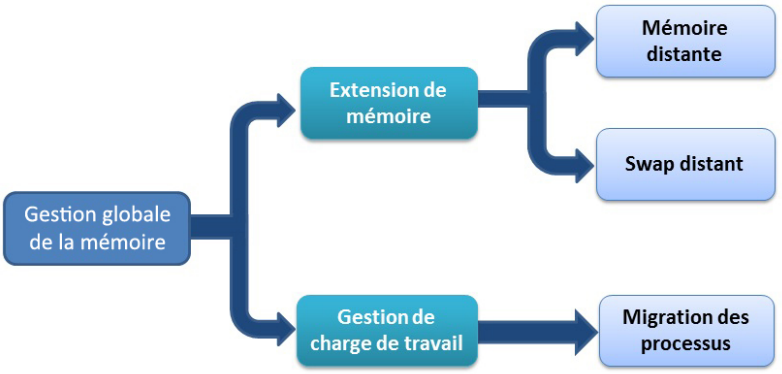

FIGURE 3.1 – Stratégies globales de gestion de mémoire

– **L'intrusivité :**
Un système global de gestion de mémoire peut être implanté à différents niveaux : système d'exploitation, application, librairie intergiciel. L'intrusivité du SGM est différente selon le niveau d'implantation. Le système peut être intrusif et exiger des modifications par rapport à l'infrastructure (par exemple en exigeant l'installation d'un nouveau système d'exploitation) ou par rapport aux applications exécutées (par exemple en exigeant la recompilation de ces dernières).

Les stratégies globales de gestion de mémoire peuvent être classées en deux catégories : les stratégies qui s'appuient sur l'extension de la mémoire principale des nœuds en s'appuyant sur les ressources disponibles sur d'autres nœuds et les stratégies qui s'appuient sur la migration des processus, telle que nous l'avons présentée à la section 2.2.3.2, pour orienter la charge de travail vers des nœuds ayant de la mémoire disponible. Chaque catégorie de stratégie peut être implantée avec des approches différentes telles que mentionnées dans la figure 3.1 [1].

Nous présentons dans les sections suivantes les différentes stratégies de gestion globale de mémoire dans une infrastructure matérielle. Nous illustrons chaque stratégie par une implantation majeure. Ces implantations de référence sont relativement anciennes. Nous les complétons en présentant également pour chaque catégorie des travaux plus récents ayant repris les mêmes principes.

1. La stratégie de réplication, que nous avons présentée au chapitre précédent (section 2.2.3.2), peut-être envisagée pour la gestion globale de la mémoire. Dans ce cas, l'application qui traite une grande charge mémoire doit être répliquée sur plusieurs nœuds afin de paralléliser le traitement et ainsi diviser la charge mémoire entre ces nœuds. Or, nous n'avons pas trouvé d'implantation de gestion globale de mémoire basée sur cette stratégie. Théoriquement, l'utilisation de cette stratégie introduit un gaspillage de mémoire puisque les applications répliquées sont installées et démarrées sur tous les nœuds. Par exemple, les processus lancés lors de l'exécution de *JBoss*(www.jboss.org) consomment environ 450 Mo. La réplication de ce dernier veut dire que cet espace doit être utilisé sur chaque nœud.

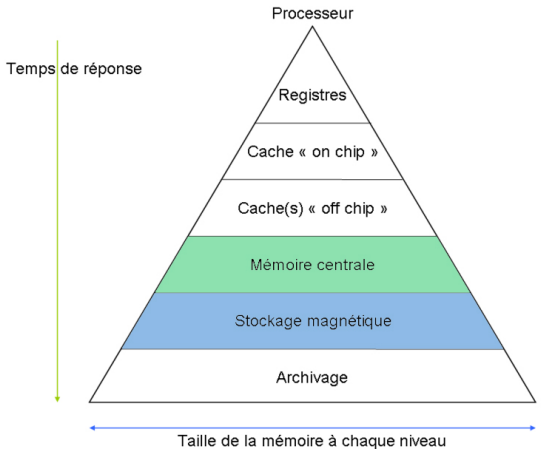

FIGURE 3.2 – Hiérarchie de mémoire

Nous actualisons ensuite par présenter des systèmes plus récents qui reprennent les mêmes principes de ces implantions de référence.

3.2 Extension de mémoire

Cette classe de stratégies consiste à étendre la mémoire d'un nœud en utilisant des ressources d'autres nœuds dans l'infrastructure (en particulier la mémoire principale). Elle se base sur la modification des techniques de mémoire virtuelle présentées précédemment pour inclure des ressources distantes.

Elle est basée sur l'architecture client-serveur. Le client est un nœud en manque de mémoire et le serveur est un nœud disposant des ressources disponibles et prêtes à être allouées aux clients. Pour améliorer les performances, cette stratégie vise à éviter les opérations de swap sur disque en augmentant la capacité de la mémoire du nœuds à travers de la mémoire disponible dans l'infrastructure. Pour cela, elles se basent sur deux faits :

- dans une infrastructure matérielle, la plupart de temps il existe des nœuds qui ont de la mémoire disponible. Plusieurs études montrent que de grandes quantités de mémoire de cluster sont presque toujours disponibles [1, 40, 5] ;
- l'émergence des réseaux à haut débit qui offrent une grande bande passante et une faible latence permettant un accès à la mémoire distante moins coûteux que l'accès à des disques locaux [1].

Les systèmes qui appartiennent à cette catégorie peuvent être classés en deux sous catégories selon le niveau d'implantation par rapport à la hiérarchie de la mémoire (figure 3.2). Ainsi, il existe des approches qui sont implantées au niveau de

la mémoire principale et d'autres qui sont implantées au niveau du stockage supplémentaire qui est utilisé en tant que support de swap.

Nous présentons ces deux sous catégories et des systèmes représentatifs dans les paragraphes suivants.

3.2.1 Mémoire partagée distribuée

La mémoire partagée distribuée est une architecture de mémoire où les mémoires de plusieurs nœuds (physiquement séparés) peuvent être traitées comme un espace d'adressage commun ce qui donne à chaque nœud dans l'infrastructure un accès à la mémoire partagée en plus de sa mémoire privée non partagée.

Cette catégorie regroupe donc des systèmes qui se basent sur la modification du mécanisme d'adressage mémoire avec comme objectif d'agrandir la mémoire principale en incluant des adresses de mémoires distantes. Grâce à cet espace supplémentaire, la mémoire du nœud est agrandie et les opérations de swap sur disque sont évitées.

Une architecture de mémoire partagée distribuée peut être mise en œuvre dans le système d'exploitation ou comme une bibliothèque de programmation. Lorsque elle est mise en œuvre dans le système d'exploitation, de tels systèmes sont transparents pour le développeur. En revanche, les architectures mises en œuvre au niveau de bibliothèque ne sont pas transparentes et les développeurs doivent généralement l'intégrer dans leurs codes et programmer différemment.

Nous présentons un exemple de cette catégorie d'approches à travers le système GMS.

3.2.1.1 Global Memory Service (GMS)

GMS [26] est un SGM dans un cluster. Il a été implanté en 1995 par l'université de Washington. GMS avait pour objectif d'améliorer la performance de tout type d'application exécutée dans le cluster. Il a été implanté au plus bas niveau du système d'exploitation *OSF/1* (niveau noyau). Le choix d'implantation au niveau du noyau du système d'exploitation permet de ne pas être intrusif au niveau des applications et donc d'être utilisable par les applications existantes sans les modifier.

Fonctionnement

Dans GMS, la mémoire de chaque nœud est divisée en deux régions : une qui stocke des pages locales (les pages du nœud en question) et une autre qui stocke les pages globales (les pages provenant d'autres nœuds). La taille de chaque région

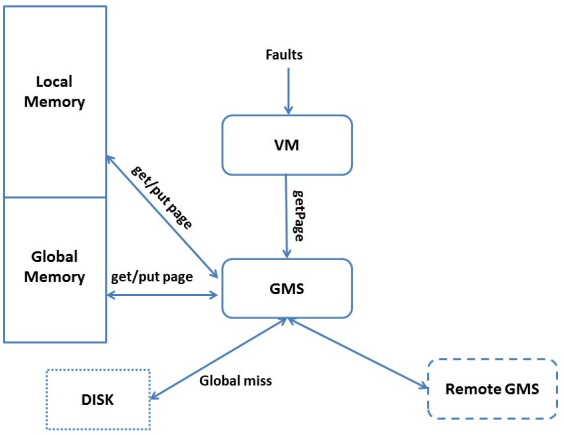

FIGURE 3.3 – GMS

change dynamiquement en réponse à la charge de travail. Ainsi, les nœuds inactifs ont une grande mémoire globale (pour servir d'autres nœuds) et les nœuds actifs ont une grande mémoire locale.

L'architecture de GMS est montrée dans la figure 3.3. Il est implanté en tant qu'un composant qui agit avec le système de mémoire virtuelle (implanté à l'espace Kernel d'*OSF*) et qui se charge de la communication avec les GMS distants. GMS réagit à chaque fois qu'il y a un défaut de page sur le nœud. Chaque défaut de page sur un nœud déclenche l'algorithme global de remplacement de page de GMS dans l'objectif de ramener la page en défaut dans la mémoire locale du nœud. Le principe de cet algorithme est comme suit :

- si la page en défaut se trouve dans une mémoire globale d'un autre nœud, elle sera échangée avec une page (victime) de la mémoire du nœud qui l'a demandé. La page victime est une page globale (s'il le nœud a une mémoire globale) ou une page locale choisie selon l'algorithme LRU (si le nœud n'a pas de mémoire globale). Cette opération permet à la page en défaut de devenir locale et augmente la taille de la mémoire locale du nœud si la page victime est une page globale ;
- si la page en défaut ne se trouve pas dans la mémoire globale, elle a été stockée sur disque car il n'avait pas de mémoire globale disponible et elle sera donc lue à partir du disque. GMS la remplace dans ce cas avec la page la plus ancienne dans le cluster (grâce à une algorithme globale d'age des pages gérée par GMS). La page est ensuite ramené à la mémoire locale du nœud comme dans le cas précédent.

Discussion

Le but de GMS est de réduire le temps moyen d'accès à la mémoire. Les expérimentations ont été faites sur un réseau AN2 ATM à 155 Mb/s. L'algorithme de GMS exploite bien la mémoire dans l'infrastructure et ne produit pas des situations

d'inefficacité. Les disques ne sont pas utilisés sauf quand il n'y a plus de mémoire globale disponible dans l'infrastructure. Étant donné que GMS est implanté au niveau de noyau de système d'exploitation, il agit au niveau des pages (dont la taille dans OSF/1 est 8Ko), ce qui fait que la mémoire est exploitée dans sa totalité (pas de fragmentation de la mémoire locale).

Malgré l'avantage d'être totalement transparent pour les applications, l'implantation de GMS au niveau de noyau de système d'exploitation OSF/1 le rend trop intrusif par rapport à l'architecture de l'infrastructure. Il faut installer le système d'exploitation modifié sur tous les nœuds. Ceci est une opération coûteuse et contraignante dans le contexte d'une infrastructure matérielle.

Les aspects d'utilisation optimale des nœuds n'étaient pas fixés en tant qu'objectif à priori. Aucun aspect de consolidation de serveurs n'a pas été envisagé dans GMS.

3.2.1.2 Autres systèmes

De nombreux projets ont exploré l'implantation d'une mémoire partagée distribuée (par exemple *IVY (université de Princeton, 1988)* [36], *SciFS (INRIA, 2002)* [19] et plus récemment *Samhita (VirginiaTech, 2011)* [52]). Ces projets mettent principalement l'accent sur les techniques de partage en réparti des données. Contrairement à ces projets, GMS insiste sur l'extension de la mémoire d'une machine en s'appuyant sur des mémoires distantes avec une gestion globale des mémoires de l'infrastructure.

3.2.2 Swap à distance

Le swap à distance est une technique qui agrège les ressources libres disponibles dans l'infrastructure pour former un espace de swap de la mémoire virtuelle. Avec cet espace de swap, le gestionnaire de mémoire virtuelle peut échanger des pages entre la mémoire locale et les ressources distantes au lieu de faire cet échange avec les disques locaux.

Dans la version la plus simple, les implantations de ces approches s'appuient sur des disques distants en tant que support de swap. Pour ce faire, elles utilisent des protocoles qui permettent de partager les ressources dans un réseau tel que NFS [2]. Grâce à ces protocoles, les clients peuvent posséder à leur disposition des espaces de stockage distants et s'en servir en tant que support de swap. Cette idée à été proposée dans un premier temps pour des machines ne disposant pas de disque local [3] ou pour améliorer les performances par rapport au swap local dans le cas où le

2. *Network File System*
3. *Diskless*

serveur distant est équipé de disques plus évolués que les disques locaux(architecture RAID [4],SAN [5],...).

Pour améliorer les performances, des systèmes qui s'appuient sur le swap sur mémoire distante ont été proposés. Dans cette catégorie, nous présentons deux systèmes implantés à deux niveaux différents : RMP et DLM.

3.2.2.1 Remote Memory Pager (RMP)

RMP [40] est un SGM dans un cluster. Il a été développé à l'institut d'informatique de la Crete en 1995. Il est l'une des premières implantations d'un système de swap à distance. Il a été implanté pour le système d'exploitation *OSF DEC*. RMP avait pour but d'améliorer la performance globale de l'infrastructure en bénéficiant de la mémoire disponible dans un cluster. Pour ce faire, il s'intéresse à minimiser des opérations de swap sur disque et de les remplacer par des swap sur mémoire distante.

Fonctionnement

Dans RMP, chaque nœud est classé selon l'utilisation de sa mémoire en client ou en serveur. Le client envoie ses demandes de swap (requêtes de lecteur-écriture de pages) vers le serveur qui utilise sa mémoire pour répondre à ces requêtes. L'implantation de RMP est distribuée. Elle est basée sur l'échange de messages entre les nœuds pour déterminer les clients et les serveurs. Un protocole de communication a été proposé. Lorsqu'un nœud devient serveur, il envoie périodiquement son état aux nœuds du cluster. Les nœuds maintiennent une liste de serveurs disponibles dans l'infrastructure. Quand un nœud devient client, il s'adresse à un des serveurs disponibles et commence à lui envoyer ses requêtes.

Le client est implanté en tant que périphérique de swap (module du noyau). Il ne demande aucune modification du système d'exploitation pour pouvoir l'utiliser. Le serveur est une application implantée au niveau utilisateur. Il accepte les connexions provenant de clients. Si le client demande une page (*pagein*), le serveur la transfère vers le client en utilisant une socket ouverte avec ce dernier. Si c'est une demande pour stocker une page (*pageout*), le serveur lit la page en question depuis la socket puis la stocke dans sa mémoire principale.

L'algorithme prévoit l'attribution dynamique des rôles de nœuds (client et serveur). Des seuils d'utilisation de mémoire sont fixés pour classer les nœuds. Un client peut devenir serveur si sa mémoire devient disponible. Un serveur peut également devenir client s'il a besoin de plus de mémoire pour ses processus locaux. Dans ce

4. *redundant array of independent disks*
5. *Storage Area Network*

cas, il met sur son disque local les pages qui ont été allouées aux clients et il notifie les autres nœuds du changement de son statut.

Discussion

Les expérimentations faites sur RMP ont été effectuées sur un cluster dont les nœuds étaient relié par un réseau Ethernet à 10 Mbps. Pour créer une charge mémoire, des outils qui produisent une charge synthétique de mémoire ont été utilisés. L'utilisation de RMP n'exige pas la modification ni du système d'exploitation ni des applications. Le système n'est donc pas intrusif.

Dans les expérimentations, on suppose que la mémoire globale disponible dans le cluster est supérieure aux besoins des applications. Elles démontrent que le swap sur mémoire distante améliore la performance par rapport au swap sur disque local. Néanmoins, l'algorithme de gestion de mémoire peut conduire à des cas d'utilisation où il n'est pas efficace. C'est le cas lorsqu'un serveur change de statut et stocke ainsi les données des clients sur son disque local sans vérifier s'il existe de la mémoire disponible sur les autres serveurs dans le cluster. Ceci peut sacrifier la performance des clients étant donné que l'accès aux données est un accès aux disques distants. Les expérimentations présentées n'évaluent pas ces surcouts.

La consolidation de serveur n'a pas été envisagée par RMP. Il ne prévoit pas de minimiser le nombre de serveurs utilisés pour fournir de la mémoire dans l'infrastructure.

3.2.2.2 Distributed Large Memory System (DLM)

DLM [42] est un SGM qui a été développé par les universités *Seikie* et *Tsukuba* en 2008. Il s'agit d'un système de swap à distance implanté au niveau utilisateur. Il a pour but d'améliorer les performances des applications en exploitant la mémoire disponible dans un cluster en évitant les opérations de swap sur disque.

Fonctionnement

DLM est implanté au niveau d'utilisateur. Il est donc indépendant du fonctionnement du système de mémoire virtuelle (géré par le noyau du système d'exploitation) et ne demande aucune modification de ce dernier.

DLM se compose d'un processus sur le nœud client permettant d'exploiter la mémoire des serveurs à travers des processus déployés sur les serveurs.

DLM propose de nouvelles fonctions de gestion de mémoire qui doivent être utilisées à la place des fonctions fournies par le système d'exploitation (par exemple *dlm-alloc* à la place de *malloc*, etc...). Ces fonctions doivent être utilisées dans le code source des applications (tel que présenté dans la figure 3.4) pour pouvoir

```
//DLM Program example : Matrix Vector Multiply
#include <stdio.h>
#define N 100000   //N:100K a: 80GB   x::800KB
dlm double a[N][N], x[N];   //DLM static declare

main(int argc, char *argv[])
{   int i,j;
   double *y;     // DLM dynamic alloc y:800KB
   y = (double *) dlm_alloc ( sizeof(double) *N );

   for(i = 0; i < N; i++)   //   Initialize a
      for(j = 0; j <N; j++)   a[i][j] = i;
   for(i = 0; i < N; i++)   x[i] = i;   // Initialize x
   for(i = 0; i < N; i++){      // multiply
       y[i]=0;                  // initialize y
       for(j = 0; j <N; j++)   y[i] += a[i][j]*x[j];
   }
   return 0;
}
```

FIGURE 3.4 – DLM

exploiter la mémoire distante. L'utilisateur peut préciser ainsi dans le code source les données qui seront hébergées dans la mémoire distante (lorsqu'il n'y a plus de mémoire locale disponible).

DLM fournit un pré-compilateur qui peut être associé à GCC. Ainsi, les programmes incluant des appels à des fonctions de DLM passent d'abord par ce pré-compilateur pour les transformer en code C standard en appelant les librairies DLM. Ils peuvent ensuite être compilés par GCC.

Lors de l'exécution de l'application compilée, les données DLM sont allouées d'abord dans la mémoire locale du nœud, s'il n'y a plus de mémoire locale disponible, elles seront hébergées dans une mémoire distante du premier serveur disponible. L'accès à cette mémoire distante est protégé et provoque un signal SIGSEGV dont le traitant récupère la page requise à partir du serveur de mémoire qui la possède. DLM échange une page locale avec la page en question.

Discussion

Les expérimentations ont été effectuées sur un cluster relié par un réseau Myrinet [12] à 10GB/s. Elles montrent une amélioration en terme du temps d'exécution de quelques applications scientifiques (*NPB, Himeno*).

DLM ne propose pas une gestion globale de la mémoire des serveurs. Il considère qu'il existe un ensemble prédéfini de serveurs de mémoire. Aucun aspect de consolidation n'a pas été envisagé. Les expérimentations présentées ne montrent pas

les surcouts qui peuvent se produire lorsqu'il y a beaucoup de clients qui utilisent le même serveur qui peut alors devenir un goulot d'étranglement. De plus, dans le cas où tous les serveurs sont chargés, DLM ne prévoit pas d'exploiter la mémoire éventuellement disponible sur d'autres nœuds clients.

L'utilisation de DLM n'exige pas de modification du système d'exploitation car il est introduit au niveau utilisateur. Par contre, il est très intrusif par rapport aux applications qui doivent être modifiées et recompilées, ce qui représente une contrainte importante.

3.2.2.3 Autres systèmes

Il existe d'autres travaux de recherches qui proposent des approches similaires de swap à distance. Nous explorons les travaux principaux dans les paragraphes suivants.

Nswap (Université de Swarthmore, 2003)

Nswap [46] est un système de swap à distance implanté pour Linux. Il propose le même modèle que RMP. Nswap classe les nœuds selon leur charge mémoire. Les clients sont équipés d'un périphérique de swap permettant d'exploiter la mémoire des serveurs à travers des processus noyaux. Nswap propose une implantation distribuée. Chaque nœud client décide de son serveur indépendamment des autres.

Étant implanté comme RMP, l'utilisation de Nswap n'exige pas la modification ni du système d'exploitation ni des applications. Le système n'est donc pas intrusif.

Les expérimentations effectuées sur Nswap avaient pour objectif de montrer l'amélioration de performance en termes de temps d'exécution. La consolidation de serveur n'a pas été envisagée par Nswap puisqu'il n'y a pas d'état global connu du cluster. Il ne prévoit pas de minimiser le nombre de serveurs utilisés pour fournir de la mémoire dans l'infrastructure.

HPBD (Université d'Ohio, 2005) :

HPBD [37] est une implantation de périphérique de swap permettant d'effectuer du swap à distance dans un réseau *InfiniBand* [48].

L'architecture d'HPBD est identique à que celle de RMP (architecture client-serveur). Le client, qui est un périphérique de swap, sert les demandes de swap provenant du noyau en communiquant avec les serveurs de mémoire à distance en utilisant les protocoles de communication d'*InfiniBand*. Le serveur est un programme qui manipule des fichiers implantés en RAM-disque [6]).

6. un disque virtuel qui utilise une partie de la mémoire centrale en tant que mémoire de masse

HPBD ne propose pas de stratégie globale de gestion de mémoire. Les expérimentations effectuées évaluent son avantage pour la performance dans un contexte statique (clients et serveurs sont prédéfinis à l'avance).

ACMS (Université de Caroline du Nord et laboratoire de recherche d'Intel, 2012) :

ACMS [56] est une implantation distribuée de gestion de mémoire basée sur le swap à distance.

Comme RMP, ACMS classe dynamiquement les nœuds dans l'infrastructure selon leur rôle de client ou serveur. Les nœuds s'échangent des messages pour s'informer de l'état mémoire. Les serveurs montent des fichiers de swap en mémoire que les clients peuvent accéder en s'appuyant sur des outils de partage de fichier.

Le protocole de communication prévoit le cas où un serveur veut réclamer la mémoire allouée aux clients. Dans ce cas, il leur envoie leurs pages ce qui peut engendrer des opérations de swap/disque malgré l'existence éventuelle de serveurs ayant de la mémoire disponible dans l'infrastructure, ce que nous considérons comme une situation d'inefficacité.

ACMS est non intrusif par rapport aux applications et à l'architecture. L'utilisation optimales des serveurs n'a pas été envisagée.

URSL (Université de Copenhague, 2010)

URSL [54] est un SGM développé implanté au niveau de l'espace utilisateur de Linux. Comme DLM, il est composé de deux éléments : un serveur de mémoire et une librairie de mémoire distante. Dans la conception de URSL, on considère qu'il existe dans l'infrastructure un ensemble fixe de serveurs dédiés pour fournir de la mémoire aux clients. Les aspects de consolidation n'ont pas été envisagés.

Contrairement à DLM, URSL n'introduit pas de nouvelles fonctions de gestion de mémoire, mais il se base sur le remplacement (par liaison dynamique) de celles qui existent déjà (tel que *malloc*, *calloc*, et *free*) à l'aide de la variable d'environnement *LD_PRELOAD*.

Puisque les serveurs sont prédéfinis, URSL présente le même inconvénient que DLM (situation d'inefficacité lorsque tous les serveurs sont chargés car il n'exploite pas la mémoire éventuellement disponible sur des nœuds clients).

L'utilisation de URSL n'exige pas de modification, ni du système d'exploitation, ni des applications.

HRP (Université de Peking et Université de Michigan, 2008)

HRP [21] est une approche pour le swap à distance pour une infrastructure virtualisée. Il est basé sur l'interception au niveau de l'hyperviseur des opérations de swap provenant des gestionnaires de mémoire des VMs. Ainsi, ces opérations sont transmises à un serveur distant prédéfini qui les traite en utilisant sa mémoire.

3.3 Migration de processus

Contrairement aux stratégies présentées précédemment, les stratégies dans cette catégories ne visent pas à modifier l'architecture de mémoire du nœud. Elles consistent plutôt à modifier l'architecture logicielle de l'infrastructure pour répondre aux besoins des applications en termes de mémoire.

Comme décrit dans la section 2.2.3.2, la migration des applications consiste à déplacer des applications qui manquent de ressources à des endroits où leurs besoins peuvent être satisfaits. Dans le cadre de la gestion de la mémoire, le déplacement a pour objectif de fournir plus de mémoire aux applications ainsi d'éviter les opérations de swap sur disque.

Grâce aux techniques de virtualisation, l'opération de migration de VM entre des nœuds selon leurs besoins en mémoire peut être envisagée et appliquée assez facilement. Néanmoins, nous n'avons pas trouvé de systèmes de gestion de mémoire basés sur ces techniques de virtualisation. Dans la pratique, la migration a été envisagée au niveau du processus dans les systèmes à image unique (Single-System Image SSI). Dans cette catégorie, nous présentons le système d'exploitation OpenMosix.

3.3.1 OpenMosix

OpenMosix [16] est un SSI, c'est-à-dire que c'est un système d'exploitation distribué qui gère le cluster dans son ensemble. Il s'agit d'une version open-source du système d'exploitation Mosix [9] qui est mis au point depuis 1981 à l'Université hébraïque de Jérusalem. [7]

L'utilisateur d'OpenMosix voit le cluster comme s'il était une seule machine. Il fournit aux utilisateurs les ressources d'une manière transparente. Les utilisateurs peuvent ouvrir une session sur n'importe quel nœud de cluster et n'ont pas besoin de connaitre l'emplacement de leurs applications.

7. la première version d'OpenMosix est sortie en 2002

Fonctionnement

OpenMosix est basé sur la répartition de la charge de travail par la migration de processus. Il déplace les processus entre les nœuds pour augmenter les performances globales des applications en leur fournissant les ressources nécessaires pour leur fonctionnement.

MOSIX offre des mécanismes globaux de gestion de mémoire hérité de Mosix [8] et implantés au niveau du noyau du système d'exploitation. Il déplace des processus exécutés dans des nœuds sans mémoire disponible vers des nœuds qui ont de la mémoire disponible pour éviter les opérations de swap sur disque. Cette opération a été prévue pour répondre à des besoins en mémoire non uniformes (des processus ayant des besoins différents de mémoire) ou des nœuds avec différentes tailles de mémoire.

Lorsque la mémoire libre d'un nœud est inférieure à un certain seuil, l'événement de migration se déclenche, le gestionnaire de mémoire choisit les processus les plus consommant de mémoire sur le nœud et les déplace vers des nœuds les moins chargés dans le cluster. Pour que la migration soit faisable, les nœuds cibles doivent disposer d'autres ressources nécessaires pour le fonctionnement des processus à migrer, telles que les ressources CPU,...

Discussion

Des outils de simulation ont été utilisés pour valider les mécanismes de gestion de mémoire d'OpenMosix. Ils montent une amélioration de performance (en termes de temps d'exécution d'application) pour des profils prédéfinis de charge mémoire. Nous constatons qu'OpenMosix n'est pas efficace dans certaines situations. En effet, vu qu'il agit au niveau des processus, OpenMosix ne peut pas traiter le manque de mémoire pour un processus qui demande une capacité de mémoire qui dépasse la mémoire maximale disponible sur un nœud. Dans ce cas le processus doit faire du swap sur disque même si la mémoire qu'il demande peut se retrouver en accumulant les mémoires disponibles de plusieurs nœuds.

Les mécanismes de gestion de la mémoire sont mis en œuvre au niveau noyau, ce qui implique une instrusivité forte au niveau de l'architecture car il faut installer OpenMosix sur tous les nœuds de l'infrastructure. OpenMosix n'est pas intrusif par rapport aux applications exécutées dans l'infrastructure. Aucune modification de celles-ci n'est requise.

Aucun aspect de consolidation de mémoire a pas été présenté dans OpenMosix.

	Inefficacité	Intrusivité	Utilisation optimale
GMS	non	Architecture : fort Application : faible	non
RMP	oui	Architecture : faible Application : faible	non
DLM	oui	Architecture : faible Application : fort	non
OpenMosix	oui	Architecture : forte Application : faible	non
URSL	oui	Architecture : faible Application : faible	non
Nswap	n/a	Architecture : faible Aplplication : faible	non
HPBD	n/a	Architecture : faible Aplplication : faible	non
ACMS	oui	Architecture : faible Aplplication : faible	non

TABLE 3.1 – Tableau récapitulatif des systèmes présentés dans l'état de l'art

3.3.2 Autres systèmes

La plupart des SSI proposent la migration des processus tel que présenté dans [38]. Par exemple, Kerrighed [45] et *OpenSSI*[8] proposent la migration des processus vers les machines ayant plus de ressources. Ils considèrent que la mémoire est une contrainte à satisfaire lors de la migration. Les processus sont donc migrés vers les machines qui peuvent satisfaire leurs besoins en mémoire.

De la même manière, nous pouvons citer *Entropy* [32]. Il s'agit d'un consolidateur de VM qui déplace les VMs dans l'objectif de trouver le meilleur emplacement des VMs (le plus petit ensemble des machines qui satisfait leurs besoins de ressources). En déplaçant les VMs, *Entropy* prend en considération la mémoire demandée par la VM.

3.4 Synthèse

Les principales caractéristiques des différents systèmes que nous venons de présenter sont résumées dans le tableau 3.1.

La gestion globale de mémoire dans une infrastructure matérielle est une problématique qui a été adressée en mettant un accent particulier sur la performance de

8. www.openssi.org

l'infrastructure : toutes les approches et les systèmes que nous avons cités avaient pour objectif d'améliorer la performance globale de l'infrastructure.

Les expérimentations montrent que cette gestion globale est effectivement bénéfique pour la performance lorsque les opérations de swap sur disque sont évitées et remplacées par des opérations mémoires. L'intrusivité varie d'une approche à l'autre. L'utilisation optimale des nœuds n'a pas été adressée par ces systèmes.

Les implantations basées sur la mémoire partagée distribuée sont trop intrusives par rapport à l'architecture de l'infrastructure. Elles sont de bas niveau. Elles impliquent généralement la modification des systèmes d'exploitation.

La migration des applications selon les besoins de ressources est une stratégie qui a été bien exploitée pour la gestion du CPU dans une infrastructure. Concernant la gestion mémoire, la migration peut être envisagée, mais dans certains cas ceci peut être impossible. Avant de faire migrer l'application, il faut garantir que le nœud cible peut assurer, en termes de ressources, tous les besoins de cette application, et pas seulement la mémoire. De plus, lorsqu'une application demande une mémoire qui dépasse la mémoire maximale fournie par un nœud, la migration devient impossible.

On remarque que l'approche la plus aboutie dans la gestion de la mémoire dans une infrastructure matérielle est le swap à distance présenté à travers le système RMP. C'est l'approche le moins intrusive par rapport à l'architecture de l'infrastructure et aux applications. Cependant, RMP ne s'intéresse pas à la gestion optimale des ressources et engendre ainsi de l'inefficacité.

Dans le chapitre suivant, nous présentons nos contributions dans la gestion de la mémoire dans une infrastructure matérielle. Notre approche, appelé *Autonomic Swap Manager* (ASM) est basée sur les techniques de swap à distance. Elle a pour objectif de répondre à nos critères de gestion de mémoire dans une infrastructure matérielle.

Deuxième partie

Contribution

Chapitre 4

Autonomic Swap Manager (ASM)

Contents

Nous avons vu au chapitre précédent les limites des systèmes de gestion globale de mémoire notamment par rapport à l'intrusivité, à la performance et à l'utilisation optimale des ressources.

Dans cette thèse, nous proposons une approche différente, basée sur les principes de la gestion de mémoire virtuelle présentés dans la section 2.3.1. Nous proposons le système *Autonomic Swap Manager* (ASM) : un système collaboratif de swap permettant la gestion globale de la mémoire dans une infrastructure matérielle. ASM permet aux systèmes informatiques distribués de collaborer afin de profiter de la mémoire disponible sur l'ensemble des nœuds de l'infrastructure pour faire face à des conditions variables de charge. Ceci permet d'éviter les opérations de swap sur disque lorsque les besoins du nœud en termes de mémoire dépassent la quantité de mémoire disponible. Il permet également de déplacer des zones de swap afin d'optimiser l'usage des ressources dans l'infrastructure.

Ce chapitre est divisé en trois parties : dans un premier temps, nous présentons les choix de conception d'ASM et son fonctionnement. Nous présentons ensuite son utilisation. Nous terminons par présenter nos choix d'implantation.

4.1 Conception

Dans cette partie, nous présentons les principes d'ASM. Nous commençons par présenter ses spécifications. Nous présentons ensuite son fonctionnement ainsi les services qu'il fournit. Puis, nous exposons en détail les politiques de gestion de mémoire proposées par ASM. Enfin, nous terminons par présenter l'architecture d'ASM ainsi que le protocole de communication utilisé.

4.1.1 Spécifications

ASM a été développé pour répondre aux besoins suivants :

- améliorer la performance d'une infrastructure distribuée en termes de mémoire en évitant l'utilisation des disques en tant que support de swap lorsqu'il y a de la mémoire disponible dans l'infrastructure. Ceci consiste donc à allouer et désallouer la mémoire dans l'infrastructure d'une manière dynamique ce qui améliore la performance surtout dans le cadre d'architectures où les nœuds ne possèdent pas de disque ;
- fournir des fonctionnalités permettant d'optimiser l'utilisation des nœuds en s'appuyant sur l'approche de consolidation. Ceci consiste à concentrer la charge mémoire sur un petit ensemble des nœuds sans pour autant dégrader la performance globale de l'infrastructure.

Pour que le système soit facilement intégrable aux infrastructures existantes, nous avons posé les contraintes suivantes :

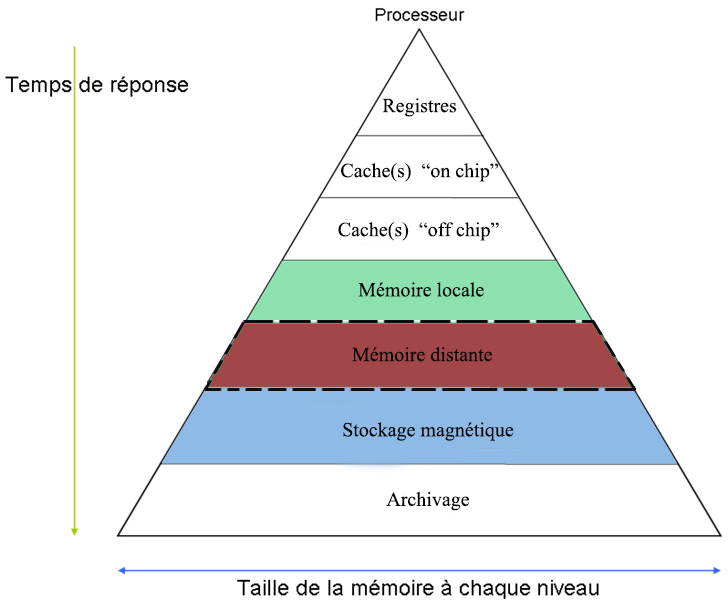

FIGURE 4.1 – Hiérarchie de mémoire incluant la mémoire distante

– le système ne doit pas être intrusif par rapport à l'infrastructure matérielle. Nous visons donc un système de gestion de mémoire pouvant s'intégrer aux systèmes d'exploitation standards ;
– le système doit être transparent aux applications qui s'exécutent dans l'infrastructure. Elles ne doivent pas être modifiées et ASM ne doit pas influencer leur fonctionnement.

En respectant ces contraintes et ces spécifications, nous cherchons à avoir un système de gestion global de mémoire répondant à tous les besoins exprimés dans l'état de l'art.

Comme nous avons vu dans le chapitre précédent, il existe plusieurs techniques qui permettent d'étendre la mémoire du nœud. Parmi ces différentes techniques, nous avons fait le choix d'utiliser celle qui se base sur le swap sur mémoire distante en tant qu'extension de mémoire locale des nœuds. Ce choix nous semble le plus adéquat dans le contexte d'une infrastructure matérielle pour les raisons suivantes :

– le swap sur mémoire distante permet d'ajouter la mémoire distante entre la mémoire principale et le disque tel que présenté dans la figure 4.1. Ainsi, toutes les techniques sophistiquées de gestion de mémoire comme la mise en cache, l'algorithme d'évacuation de pages, etc ... qui ont évolué dans le passé pour optimiser les performances de la gestion locale de mémoire ne sont pas modifiés et fonctionnent normalement ;

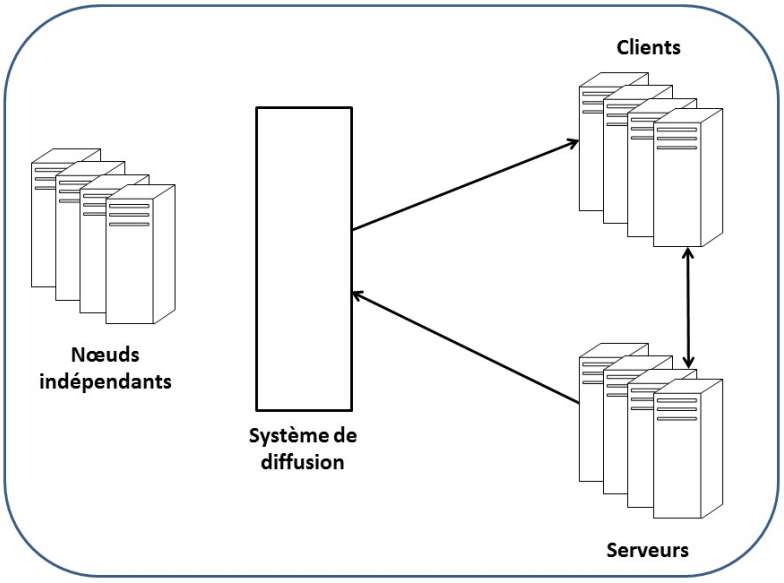

FIGURE 4.2 – Vu générale de composant d'ASM

- la modification de support du swap n'implique pas de mettre à jour le système d'exploitation qui est une opération coûteuse surtout au sein d'une infrastructure matérielle. Implanter un nouveau support de swap peut être fait sans interrompre le fonctionnement du système. Nous en discutons plus en détail dans les sections suivantes ;
- le swap à distance permet aux processus de prendre avantage de la mémoire distante de manière transparente car elle est gérée au niveau du système d'exploitation.

Nous proposons donc un service global de gestion de swap basé sur la collaboration de mémoire dans une infrastructure matérielle. Il permet aux nœuds de découvrir de façon dynamique, allouer et désallouer la mémoire à distance selon les besoins en mémoire locale. Notre approche est basée sur une architecture client-serveur. Nous présentons son fonctionnement général dans les paragraphes suivants.

4.1.2 Fonctionnement général d'ASM

La figure 4.2 présente une vue globale de l'infrastructure lors de l'utilisation d'ASM. Nous pouvons voir que les nœuds sont classés en trois catégories en fonction de leur utilisation de mémoire : client, serveur et indépendant.

Les clients sont des nœuds qui exécutent des applications avec une forte demande de mémoire et qui peuvent avoir besoin d'espace mémoire supplémentaire. Pour cela, ils s'appuient sur des serveurs qui sont des nœuds possédant une grande quantité de mémoire libre pouvant potentiellement être allouée aux clients. Les nœuds indépen-

FIGURE 4.3 – Classification de nœuds

dants sont des nœuds qui ont un niveau intermédiaire d'utilisation de mémoire. Ils n'offrent donc pas de mémoire et qu'ils n'ont pas besoin de mémoire supplémentaire.

Une fois le type de nœud identifié, il communique avec les autres nœuds à travers un service global d'échange de message fourni par ASM et basé sur une architecture publication/souscription [1].

Le type du nœud peut changer dynamiquement suite au changement de sa charge mémoire qui peut varier selon les applications exécutées sur le nœud et les phases d'exécution de celles-ci. Nous détaillons cette opération dans le paragraphe suivant.

4.1.2.1 Classification des nœuds

ASM se base sur la charge locale de mémoire [2] des nœuds pour pouvoir les classer en client, serveur ou indépendant. Pour ce faire, ASM définit deux seuils contrôlant le taux d'occupation de la mémoire vive du nœud tel que présenté dans la figure 4.3. Ainsi :

- Lorsque l'utilisation de la mémoire du nœud est inférieure au seuil MIN (qui indique une très faible demande de mémoire locale), le nœud est classé comme un serveur de mémoire ;
- Si l'utilisation de la mémoire du nœud est supérieure au seuil MAX (indiquant une utilisation potentiellement élevée de mémoire) le nœud devient un client de mémoire ;
- Si sa charge mémoire reste entre MIN et MAX le nœud devient un nœud indépendant.

La classification d'un nœud est basée sur l'utilisation de mémoire vive et non sur les opérations qu'il effectue. Ainsi, lorsqu'un nœud devient serveur, ceci signifie qu'il peut offrir de la mémoire aux clients sans forcément indiquer que cette mémoire est effectivement utilisée par des clients. De la même façon, la classification d'un nœud

1. Pour simplifier, nous utiliserons le terme *PubSub* (de l'anglais *Publish-Subscribe*) pour designer ce service
2. mémoire utilisée par les processus locaux du nœud.

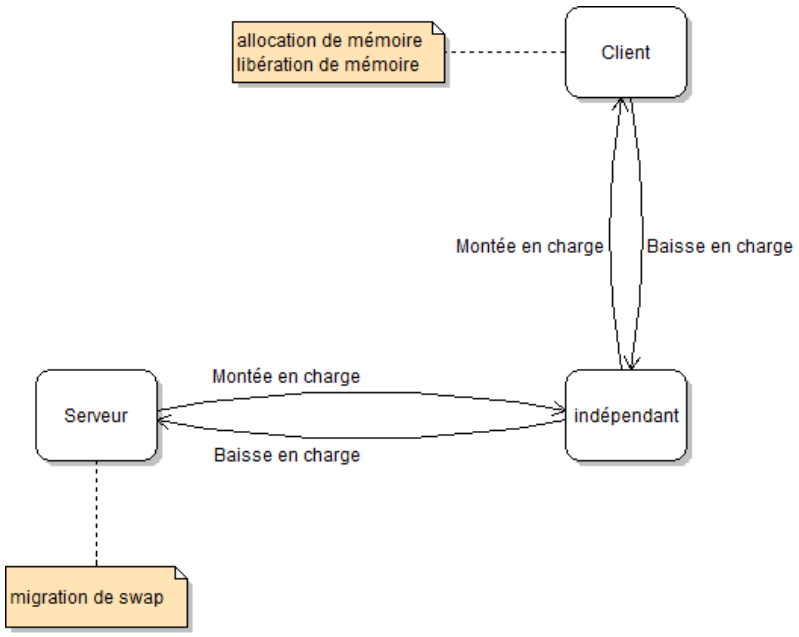

FIGURE 4.4 – Actions

en client signifie qu'il peut avoir besoin de mémoire supplémentaire. Les actions sont ensuite effectuées selon le type du nœud (client, serveur ou indépendant) et selon sa charge mémoire telle que présentée dans la figure 4.4. Nous détaillons ces opérations dans les paragraphes suivants.

4.1.2.2 Fonctionnement des clients

La classification du nœud en tant que client signifie qu'il a une charge mémoire relativement élevée et qu'il aura besoin de mémoire supplémentaire si ça charge mémoire continue à monter. ASM définit un autre seuil (HIGH) qui indique un taux trop élevé d'occupation de mémoire centrale du nœud [3] et que la mémoire doit être étendue avant qu'il commence à swapper. Une fois que la charge mémoire du nœud atteint ce seuil, ASM étend sa mémoire par un service de swap à distance en s'appuyant sur la mémoire fournie par les serveurs tel que présenté dans la figure 4.5. Ainsi, le système d'exploitation peut utiliser cette mémoire fournie par les serveurs en tant que zone de swap. Il effectue des opérations *swapin* et *swapout* des pages mémoires sur ces zones de swap sans aucune influence de la part d'ASM.

Si la charge mémoire du client baisse, il peut libérer la mémoire qu'il a alloué auprès des serveurs. ASM propose deux politiques pour effectuer cette opération. Elles seront présentées dans le paragraphe 4.1.4.2.

3. Par mémoire centrale, nous voulons dire la mémoire vive ainsi le swap du nœud.

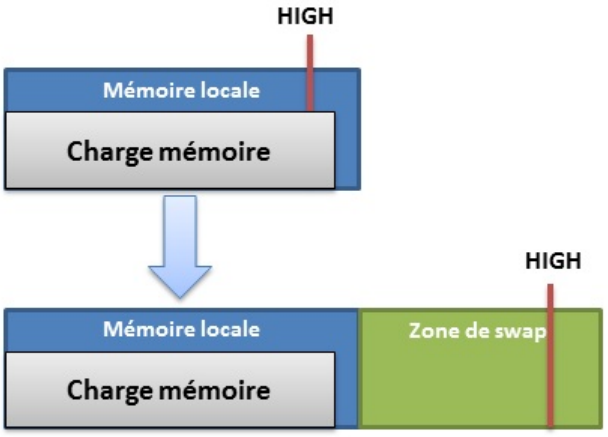

FIGURE 4.5 – Étendre la mémoire de client

4.1.2.3 Fonctionnement des serveurs

Le serveur fournit une partie de sa mémoire aux clients. Sa mémoire est donc divisée en deux parties : une qui est allouée aux clients (mémoire hébergée) et l'autre est réservée pour ses processus locaux (mémoire locale).

L'allocation de mémoire hébergée est faite à la demande des clients. Ainsi, la mémoire hébergée est divisée en plusieurs zones (des chunks) qui sont utilisées par les clients en tant que support de swap.

Afin de ne pas pénaliser le fonctionnement du serveur, il doit garder un minimum de mémoire locale disponible. Pour ce faire, ASM définit un seuil (LOW) représentant la quantité minimale de mémoire locale qui doit être être disponible. Tel que présenté dans la figure 4.6, le serveur augmente sa mémoire locale en réclamant la mémoire hébergée des clients si sa charge de mémoire locale dépasse ce seuil. Nous détaillons les politiques contrôlant cette opération dans le paragraphe 4.1.4.3.

4.1.3 Services fournis par ASM

Dans les paragraphes suivants, nous présentons le fonctionnement des deux services fournis par ASM.

4.1.3.1 Service publication/souscription

Le service de *PubSub* est un service global assuré par ASM pour fournir un système d'échange de message entre les nœuds. Ce service permet aux nœuds d'échanger des messages sur l'état de la mémoire ce qui permet de contrôler les opérations d'at-

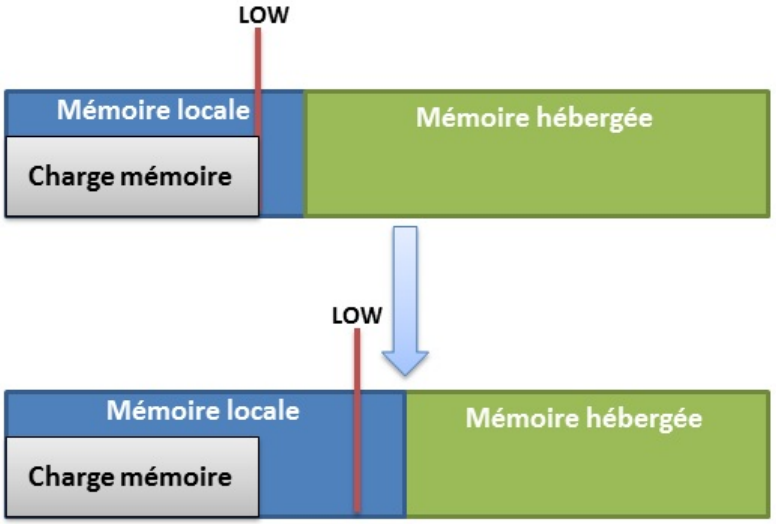

FIGURE 4.6 – récupération de mémoire de serveur

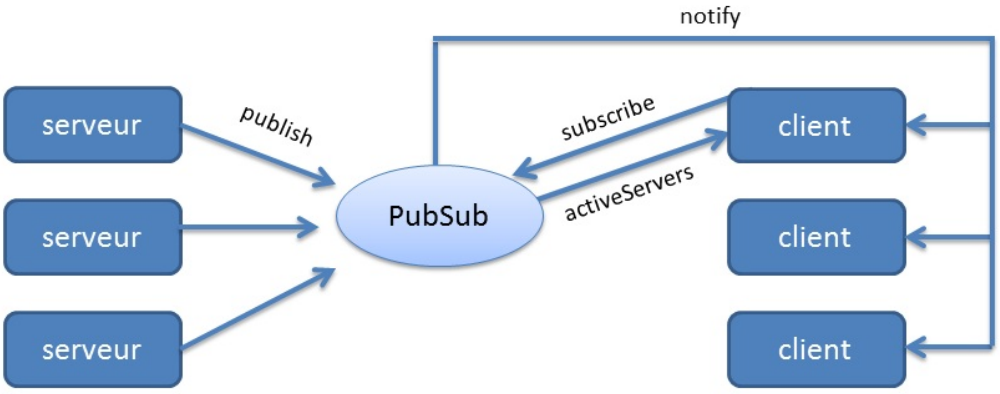

FIGURE 4.7 – Service PubSub

tribution de mémoire dans l'infrastructure matérielle. En particulier, il permet aux nœuds d'avoir accès aux événements concernant la disponibilité de mémoire dans l'infrastructure.

Ce service est composé d'un système d'abonnement et de diffusion tel que présenté dans la figure 4.7. Le système fournit deux interfaces : une est utilisée par les nœuds qui publient des informations (les *publishers*) pour permettre la diffusion d'informations, et l'autre est utilisée par les nœuds qui veulent recevoir les informations publiées (les *subscribers*) pour qu'ils puissent adhérer au service.

Lorsqu'un nœud devient client, il s'abonne au service pour connaître l'état des serveurs disponibles dans l'infrastructure. Les serveurs publient les informations concernant leurs états de mémoire à tous les clients abonnés au service. Ces in-

formations comprennent la quantité de mémoire qu'ils offrent ainsi le nombre de clients qu'ils hébergent.

La publication des informations (provenant des serveurs) est contrôlée par un paramètre (*ServerStep*) qui détermine le seuil de variation de mémoire disponible à partir duquel le serveur doit publier. S'il dispose par exemple d'une mémoire disponible de 1000 Mo et qu'on a assigné 50 Mo à *ServerStep*, le serveur publie lorsque sa mémoire disponible est de 1050 Mo ou de 950 Mo. Ceci permet aux administrateurs de l'infrastructure de faire des compromis entre la précision des états des serveurs aperçus par les clients et le nombre de messages envoyés dans le réseau. Affecter une petite valeur (par exemple 1 Mo) permet d'informer en permanence de l'état de mémoire des serveurs mais cela peut engendrer beaucoup de messages transmis sur le réseau. Par contre lorsqu'on affecte une grande valeur (par exemple 700 Mo), les clients vont avoir une vue partielle de l'état de la mémoire de serveurs, mais avec un nombre de messages moins élevé.

Les clients abonnés au service gèrent les informations reçues des serveurs pour construire une liste de serveurs disponibles et l'état de leur mémoire. Ainsi, lorsqu'ils ont besoin, ils peuvent choisir un serveur pour utiliser sa mémoire en tant que support de swap. ASM propose deux politiques de choix d'un serveur selon l'objectif déterminé par l'administrateur. Nous les présentons dans la section 4.1.4.1.

Une fois le choix fait, le client et le serveur sont configurés pour permettre les opérations de swap à distance. Nous présentons ce service dans le paragraphe suivant.

4.1.3.2 Service de swap à distance

Le service de swap à distance représente le cœur d'ASM. Il s'agit des mécanismes qui permettent d'étendre la mémoire du client en s'appuyant sur la mémoire du serveur. Ils sont assurés par des éléments d'ASM implantés au niveau du client et d'autres implantés au niveau du serveur tels que présentés dans la figure 4.8.

Au niveau du client, ASM déploie un périphérique de type block [4] qui va être utilisé en tant que support de swap. Ce périphérique a été conçu pour être installé sans avoir à modifier le noyau de système d'exploitation. Il n'agit sur aucune fonctionnalité du système de gestion de mémoire virtuelle. Toutes les opérations d'évacuation des pages et de remise en cache sont exécutées normalement.

Au niveau du serveur, ASM fournit un service d'exportation de mémoire. Il fournit des interfaces sur le réseau permettant au client d'effectuer des opérations de lectures et d'écriture sur la mémoire du serveur.

4. *Block device*

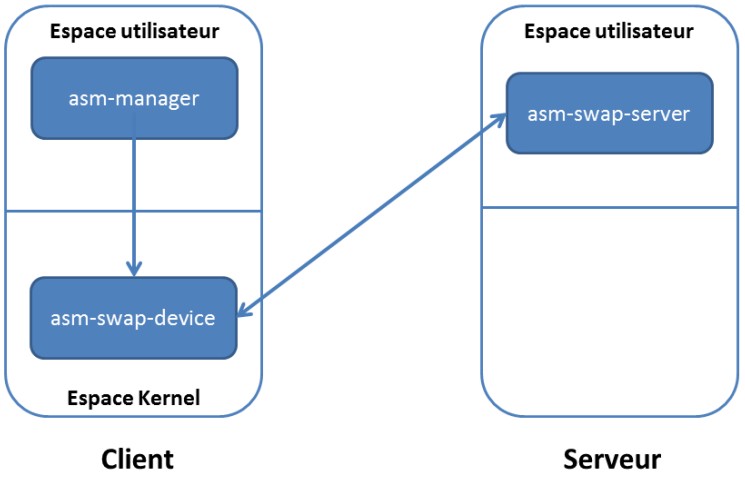

FIGURE 4.8 – ASM Core

Le client choisit un serveur parmi la liste des serveurs disponibles dans l'infrastructure reçue par le système *PubSub*. ASM configure le client et le serveur pour assurer la communication. Ainsi, quand la mémoire du client est saturée, le gestionnaire de mémoire (non modifié dans Linux) utilise le périphérique fourni par ASM pour évacuer les anciennes pages et libérer ainsi la mémoire principale. Le périphérique fournit des interfaces standards (les mêmes que celles d'un autre périphérique de swap) qui permettent d'effectuer des opérations lectures ou écriture à la demande de gestionnaire de mémoire. Chacune de ces opérations sera transmise par le réseau au serveur qui traite ces opérations sur la mémoire réservée aux clients.

4.1.4 Politique globale de gestion de mémoire

Dans cette section, nous détaillons les politiques globales de gestion de mémoire fournies par ASM. Nous détaillons en particulier les politiques qui concernent l'attribution de mémoire aux clients et la libération de mémoire des serveurs.

4.1.4.1 Attribution de mémoire aux clients

Le service *PubSub* tient informé les clients des serveurs disponibles dans l'infrastructure et la mémoire que chaque serveur peut fournir. À partir de cette information, les clients peuvent utiliser cette mémoire par le biais du périphérique de swap. Comme dit précédemment, ce périphérique doit être configuré avec un serveur avant qu'il ne soit utilisé. Le client donc doit faire le choix d'un serveur parmi les serveurs disponibles pour utiliser sa mémoire.

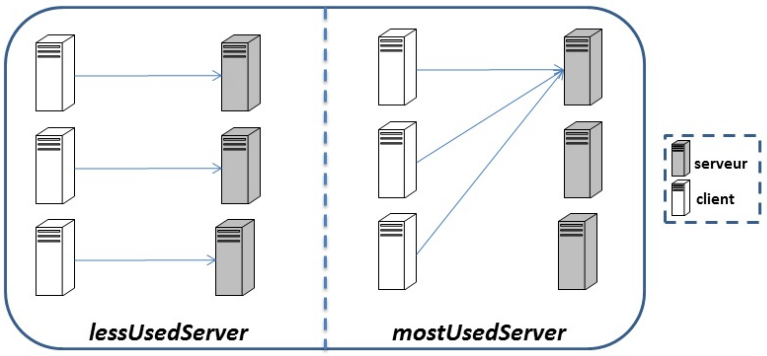

FIGURE 4.9 – Politiques d'allocation de mémoire

ASM propose deux stratégies de choix de serveur. La figure 4.9 présente un exemple d'utilisation de ces stratégies. Ainsi :

- *mostUsedServer* : en utilisant cette stratégie, le client s'adresse aux serveurs qui hébergent déjà de la mémoire d'autres clients et qui ont encore de la mémoire disponible. Si aucun serveur utilisé ne peut satisfaire la demande du client, alors le client s'adresse à un nouveau serveur. Cette stratégie valorise la minimisation du nombre de serveurs utilisés dans l'infrastructure en consolidant la charge mémoire sur le plus petit ensemble des serveurs actifs ;
- *lessUsedServer* : en utilisant cette stratégie, le client s'adresse aux serveurs qui ne sont pas utilisés ou ceux qui hébergent le moins de clients. Cette stratégie valorise le parallélisme des entrées/sorties au niveau des serveurs et évite les goulots d'étranglement.

Une fois le choix d'un serveur fait, le client échange des messages avec le serveur pour configurer le périphérique de swap. Il instancie un nouveau périphérique puis il le configure pour qu'il communique avec le serveur choisi pour toute opération de lecture ou d'écriture provenant du système de gestion de mémoire virtuelle. Il informe enfin le système de gestion de mémoire virtuelle de l'existence d'un nouvel espace de swap.

Cette opération peut être refaite à chaque fois où la mémoire du client dépasse le seuil maximum et tant qu'il y a encore des serveurs disponibles. S'il n'y a plus de mémoire disponible dans l'infrastructure, deux choix sont possibles :

- si le nœud dispose d'un disque local, le gestionnaire de mémoire fait des opérations de swap sur disque. ASM n'intervient pas ;
- si le nœud ne dispose pas de disque, ASM peut être utilisé pour effectuer le swap sur un disque distant. Cette opération peut être intéressante lors de l'utilisation de disques rapides tels que les disques SSD [5].

5. *solid-state drive*

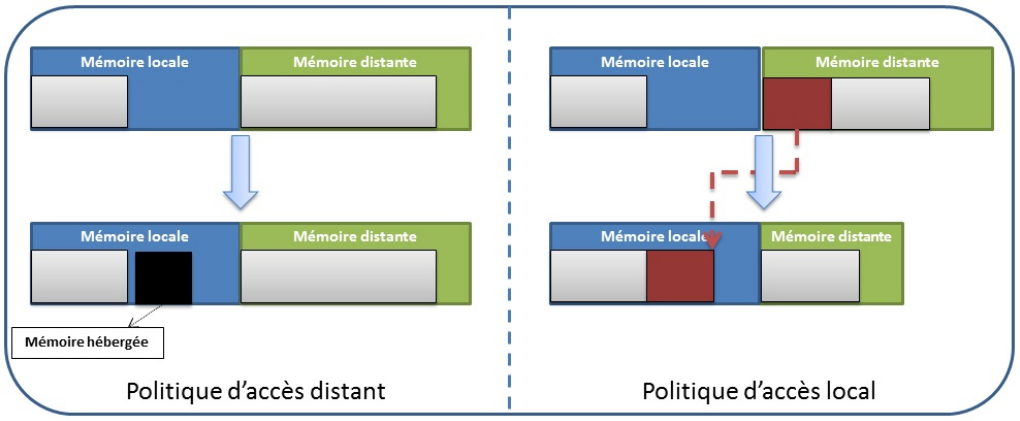

FIGURE 4.10 – Politiques de libération de mémoire

4.1.4.2 Libération de mémoire

Le client peut utiliser la mémoire allouée par les serveurs tant qu'il en a besoin. Si sa mémoire principale devient disponible suite à la baisse de charge mémoire, il faut faire bénéficier d'autres clients de la mémoire disponible.

Le gestionnaire de mémoire virtuelle du système d'exploitation n'a pas forcément intérêt à chercher les pages qui ont été swappées sauf à la demande des processus. Cette situation implique qu'il peut y avoir des clients qui ont de la mémoire locale disponible et qui utilisent quand même de la mémoire distante.

Pour faire face à cette situation, ASM propose deux politiques dans l'objectif de faire bénéficier d'autres clients de la mémoire disponible :

– la politique d'accès local : le principe de cette politique consiste à favoriser l'accès à des données locales au niveau du client. En appliquant cette politique, le client libère la mémoire distante et la rend disponible dès qu'il a de la mémoire locale disponible. Ceci implique la migration des données à chaque fois qu'un client a de la mémoire disponible. Pour ne pas prendre des décisions incohérentes et éviter les déplacements récursifs, ASM doit valider la condition suivante avant d'effectuer la migration :

$$ChunkSize + MemorySize < HIGH$$

ou : *ChunkSize* est la taille de la zone de swap à rapatrier, *MemorySize* est la taille de la mémoire locale utilisée sur le client et *HIGH* est le seuil calculé en fonction de la taille de la mémoire centrale après la migration.
La figure 4.10 (côté droit) montre un exemple de cette politique. Nous pouvons constater sur cette figure que la charge mémoire a été transférée de la mémoire distante vers la mémoire locale du nœud et que cette mémoire distante a été libérée ;

– la politique d'accès distant : le principe de cette politique consiste à ne pas chercher les pages qui ont été stockées à distance sauf quand le client les demande. L'utilisation de cette politique revient à avoir un état hybride. Ainsi, si le client a de la mémoire disponible et utilise quand même de la mémoire distante, il emprunte le comportement du serveur en publiant qu'il dispose de mémoire disponible. Ainsi, sa mémoire peut être utilisée par d'autres clients bien qu'il ait encore des pages stockées à distance.
La figure 4.10 (côté gauche) montre un exemple de cette politique. Nous constatons que la mémoire distante du client n'a pas été changée, et que la mémoire disponible chez le client a été utilisée par un autre client (représenté par un carré noir)

Dans la version actuelle de notre prototype, nous avons implanté uniquement la politique d'accès local.

4.1.4.3 Récupération de mémoire

Comme nous l'avons expliqué dans le paragraphe 4.1.2, ASM considère que tout nœud peut devenir serveur lorsqu'il dispose d'assez de mémoire disponible. Pour ne pas pénaliser les serveurs lorsqu'ils ont besoin d'utiliser leur mémoire locale, ils gardent un minimum de mémoire disponible pour leurs processus locaux. Un serveur peut décider de récupérer la mémoire allouée aux clients. Pour ce faire, il fait appel aux autres serveurs dans l'infrastructure pour trouver un serveur qui peut héberger la mémoire de ses clients.

Quand un serveur est trouvé, il commence à lui envoyer les données du plus ancien client hébergé chez lui. La récupération de la mémoire du serveur peut être faite de deux façons :

– *freeWhenFinished* : en utilisant cette politique, la mémoire allouée au client peut être libérée une fois que le serveur termine d'envoyer la totalité des pages hébergée chez lui vers le nouveau serveur. Si le client demande des pages pendant la transition, elles seront accessibles normalement depuis l'ancien serveur sans coût supplémentaire pour le client, mais ceci peut engendrer des coûts mémoire au niveau du serveur (redondance temporaire des données pendant la transition) ;
– *freeProgressively* : en utilisant cette politique, le serveur récupère l'emplacement de la page dès qu'elle est envoyée au nouveau serveur. Dans ce cas, si pendant la transition, le client demande une page qui a été transférée, le serveur transfère cette demande vers le nouveau serveur qui se charge de répondre au client. Ceci engendre des couts au niveau des clients, mais un bénéfice pour le serveur.

Une fois la migration terminée, le serveur notifie le client du nouvel emplacement de ses pages.

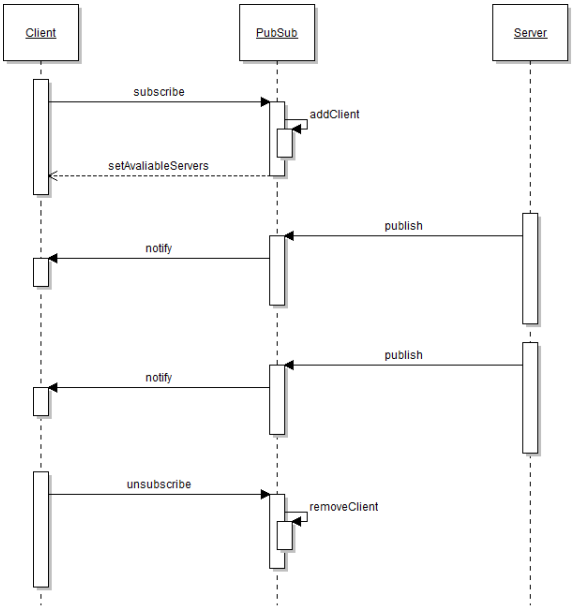

FIGURE 4.11 – Suscription/publication

4.1.5 Protocole

Comme dit précédemment, le fonctionnement d'ASM est basé sur l'échange de messages entre les nœuds gérés par le service d'abonnement. Nous avons défini un protocole de communication permettant d'organiser les opérations traitées par ASM. Le protocole permet aux nœuds d'échanger des informations sur leur disponibilité et leur besoin en termes de mémoire.

Les messages définis par le protocole sont décrits dans les paragraphes suivants. Nous présentons les scénarios d'exécution et les messages échangés pour chaque opération gérée par ASM.

4.1.5.1 Cycle de vie global

La figure 4.11 présente les suites de messages échangés entre le service *PubSub*, les clients et les serveurs lors des opérations de base du service *PubSub* (la publication et la souscription). Le scénario est le suivant :

1. lorsqu'un nœud devient client car son taux d'utilisation de la mémoire dépasse le seuil MAX, il s'abonne auprès du service *PubSub* en envoyant le message *subscribe* au port dédié à la souscription du service *PubSub*. Ce dernier ajoute le nœud dans la liste de clients actifs. Il lui envoie en retour le message *setAvaliableServers* qui contient l'état actuel des serveurs actifs dans l'infrastructure pour lui permettre de contacter des serveurs lorsqu'il y a besoin ;

2. lorsqu'un nœud devient serveur, il envoie au service *PubSub* le message *publish* en passant son adresse IP et la mémoire disponible qu'il peut fournir aux clients. Le service *PubSub* envoie le message *notify* à tous les clients abonnés au service. Ce message permet aux clients de prendre connaissance de l'existence du nouveau serveur et de la quantité de mémoire qu'il peut fournir ;
3. le serveur envoie une mise à jour de son état chaque fois que son utilisation de mémoire varie d'une valeur équivalente du paramètre *PubStep* tel que nous l'avons expliqué dans la section 4.1.3.1. Le service *PubSub* réagit de la même façon en envoyant le message *notifyClient* aux clients pour qu'ils mettent à jour l'information concernant le serveur en question ;
4. quand le client n'a plus besoin de mémoire supplémentaire (charge mémoire inférieure à MAX et pas d'utilisation de mémoire distante), il devient indépendant et il envoie le message *unsubscribe* pour ne plus recevoir les informations concernant les serveurs actifs , le service *PubSub* retire le client de sa liste de diffusion ;
5. si la quantité de mémoire utilisée localement par un serveur augmente et qu'il redevient un nœud indépendant, il envoie au service *PubSub* le message *publish* en indiquant qu'il ne dispose plus de mémoire à fournir aux clients. Le service *PubSub* notifie les clients et le serveur n'envoie plus rien.

Un serveur peut à tout moment recevoir des demandes provenant des clients pour qu'il alloue une partie de sa mémoire. Nous détaillons cette opération dans le paragraphe suivant. Il peut également récupérer la mémoire allouée aux clients quand il en a besoin. Nous détaillons cette opération dans la section 4.1.5.3.

4.1.5.2 Étendre la mémoire d'un client

Lorsque l'état de la mémoire d'un client devient critique (sa charge mémoire atteint HIGH), ASM réagit en choisissant un serveur disponible pour utiliser une partie de sa mémoire en tant que support de swap. Le protocole de communication est présenté dans la figure 4.12. Les messages sont échangés comme suit :

1. le client envoie le message *bookMemory* au serveur choisi en précisant son adresse IP et la taille de mémoire demandée ;
2. le serveur envoie le message (*publish*) au service *PubSub* pour mettre à jour la quantité de mémoire dont il dispose ;
3. il envoie au client le port d'écoute qui permet au client d'utiliser la mémoire allouée ;
4. le client peut alors utiliser la mémoire fournie par le serveur en tant que support de swap. Toute opération de lecture/écriture effectuée sur ce support par le système de gestion de mémoire virtuelle sera transmise au serveur via le message *doRead* ou *doWrite*.

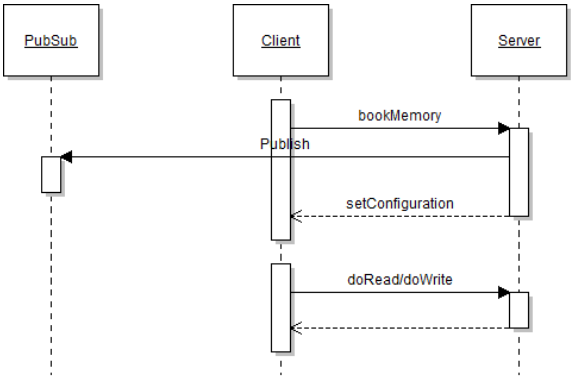

FIGURE 4.12 – Étendre la mémoire de client

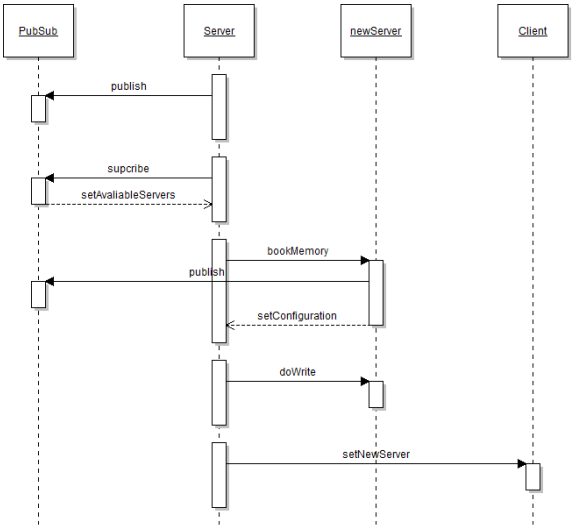

FIGURE 4.13 – Vu générale de composant d'ASM

S'il n'y a pas de serveur dans l'infrastructure qui peut assurer la totalité de la mémoire demandée par le client, ce dernier s'adresse à plusieurs serveurs de la même façon pour satisfaire ses besoins.

4.1.5.3 Récupération de mémoire

Pour la gestion optimale de performance, ASM prévoit les cas où un serveur veut récupérer la mémoire allouée aux clients pour ses propres processus. Ceci se produit quand la mémoire locale du serveur atteint un seuil minimum de mémoire disponible (LOW). Cette opération est présentée dans la figure 4.13. Les suites des messages échangés sont comme suit :

1. le serveur envoie le message *publish* au service *PubSub* pour informer les clients abonnés qu'il ne prête plus de mémoire ;

2. le serveur s'abonne en tant que client auprès du service *PubSub* en envoyant le message *subscribe*. Il a en retour la liste de serveurs disponibles dans l'infrastructure. Ainsi, il peut choisir un nouveau serveur pouvant héberger la mémoire du client ;
3. le serveur contacte le nouveau serveur pour réserver de la mémoire comme s'il était un nouveau client. Le nouveau serveur bloque la mémoire demandée et publie son nouvel état aux clients par le biais du service *PubSub* ;
4. le serveur commence ensuite à migrer les données de sa mémoire vers le nouveau serveur en envoyant des messages (*doWrite*) au nouveau serveur en suivant la politique choisie ;
5. une fois cette migration terminée, le serveur notifie le client du nouvel emplacement de ses données en lui envoyant le message *setNewServer*.

4.2 Utilisation

ASM a été implanté pour être générique et utilisable sans modifier l'infrastructure matérielle. Il a été implanté en utilisant le langage C pour le système d'exploitation Linux. Il ne demande aucune modification de Linux et il peut être installé sur des nœuds sans avoir à les redémarrer ou à interrompre leur fonctionnement.

Au sein de l'infrastructure matérielle, il faut désigner un nœud sur lequel le service *PubSub* sera installé. Nous considérons que ce service doit être installé et démarré avant le démarrage d'ASM au niveau des nœuds pour permettre la communication entre eux. Nous détaillons dans les paragraphes suivants les procédures d'installation, configuration et démarrage d'ASM

4.2.1 Configuration

La configuration d'ASM se fait à l'aide d'un fichier texte contenant les noms des paramètres et les valeurs associées. Les principaux paramètres sont :

- ***PubSubNode*** : ce paramètre indique l'adresse IP du nœud où est installé le service *PubSub*. Il sera utilisé par les noeuds lorsqu'ils deviennent clients ou serveurs ;
- ***PubSubClientPort*** : ce paramètre indique le port permettant la souscription des clients sur le nœud *PubSubNode* ;
- ***PubSubServerPort*** : ce paramètre indique le port permettant aux serveurs de diffuser leur information sur le nœud *PubSubNode* ;
- ***MAX, MIN, HIGH, LOW*** : ces paramètres indiquent les seuils qui contrôlent les opérations de la classification des nœuds, de l'allocation et de la libération de mémoire tel que nous l'avons expliqué dans la section 4.1.2 ;
- ***MemoryChunk*** : ce paramètre indique la taille de mémoire à allouer lorsqu'un un client sollicite un serveur. Pour faciliter l'allocation de mémoire, nous considérons que les serveurs allouent à chaque fois une taille fixe de mémoire aux clients. Étant donné qu'ASM n'est pas intrusif par rapport aux applications, les besoins réels de celles dernières ne sont pas connus à priori. Ce paramètre peut changer dynamiquement lorsque la mémoire disponible du serveur est inférieure à *MemoryChunk* ;
- ***ServerStep*** : ce paramètre contrôle la fréquence de publication d'information des parts des serveurs tel que nous l'avons expliqué dans la section 4.1.3.1 ;
- ***memoryAllocationPolicy, freeingMemoryPolicy, memoryReclaimPolicy*** : ces paramètres indiquent les politiques de gestion de mémoire telles que nous les avons expliquées dans le paragraphe 4.1.4.

4.2.2 Installation

ASM doit être déployé, configuré, installé et démarré sur tous les nœuds de l'infrastructure. Cette opération est difficilement faisable manuellement lors d'une utilisation dans une grande infrastructure. Pour faciliter cette tâche, nous avons utilisé un système appelé TUNe[15]. Il s'agit d'un système d'administration autonome qui permet depuis un nœud central de déployer une application sur différents nœuds. Il permet aussi de reconfigurer cette application suite à une panne ou à des événements prédéfinis. Un avantage de TUNe est qu'il fournit des formalismes de haut niveau (des profils UML) pour toutes les étapes de déploiement et de reconfiguration ce qui rend son utilisation aisée pour l'administrateur de l'infrastructure.

Pour installer ASM en s'appuyant sur TUNe, nous avons défini des diagrammes UML précisant :

- un diagramme de déploiement : il indique les unités d'ASM à installer sur chaque nœud et les liens entre eux ;
- un diagramme de nœuds : il indique les nœuds sur lesquels ASM sera installé, et les paramètres généraux de déploiement (répertoire, protocole d'accès aux nœuds ...) ;
- un diagramme de reconfiguration : il indique les procédures pour redéployer ASM s'il tombe en panne sur un nœud.

TUNe peut être lancé en précisant le nom de projet à déployer (ASM). Il procède comme suit :

- copie le fichier d'archive d'ASM sur chaque nœud ;
- décompression de l'archive sur les nœuds distants ;
- configuration d'ASM sur chaque nœud ;
- démarrage d'ASM sur chaque nœud.

L'installation d'ASM sur les nœuds exige avoir les droits d'accès du super utilisateur (*root*) notamment pour l'installation et la désinstallation de périphérique de swap. En vue de renforcer la sécurité, nous utilisons la commande *sudo* qui permet d'exécuter les commandes d'installation/désinstallation des périphériques (les commandes *insmod/rmmod*) avec les droits *root*.

Une fois l'installation et le démarrage d'ASM terminés, il n'est plus nécessaire d'intervenir sauf dans le cas ou on veut installer ASM sur de nouveaux nœuds.

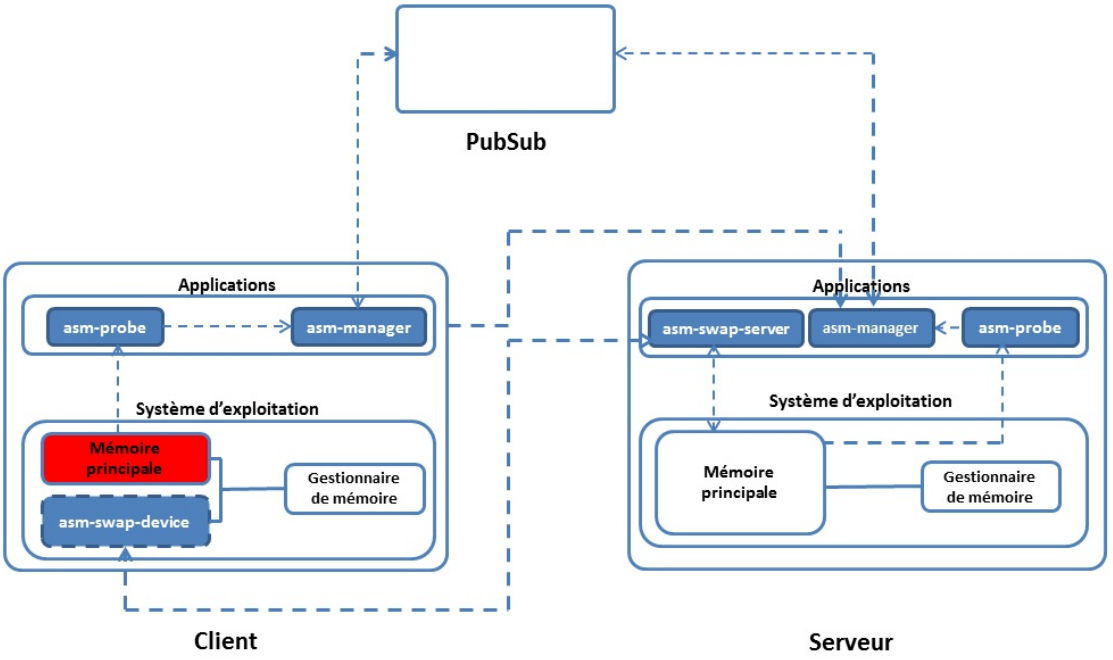

FIGURE 4.14 – Architecture interne d'ASM

4.3 Implantation

Dans cette partie, nous présentons nos choix d'implantation pour notre approche ASM.

4.3.1 Vue détaillée

L'architecture interne d'ASM est présentée dans la figure 4.14. Cette figure montre une vue logique d'ASM distinguant les clients et les serveurs. Les boites bleues représentent les unités d'ASM déployées sur chaque nœud. Nous y trouvons :

- *asm-probe* : cette unité est utilisée pour récupérer les informations d'utilisation réelle de mémoire locale du nœud. Ces informations sont utilisées pour la prise de décision concernant la classification du nœud ou les opérations d'allocation de mémoire ;
- *asm-manager* : cette unité est l'unité principale d'ASM. Elle reçoit les informations provenant d'*asm-probe* et les utilise pour la prise de décisions. Elle gère également la communication avec le système *PubSub* et avec les autres nœuds dans l'infrastructure ;
- *asm-swap-device* : cette unité est utilisée au niveau du client. Il s'agit du périphérique de swap qui peut être configuré pour communiquer avec les serveurs ;

– *asm-swap-server* : cette unité est utilisée au niveau du serveur. Elle permet de fournir la mémoire du serveur pour qu'elle soit utilisée à distance par *asm-swap-device*.

Ces unités d'ASM permettent d'assurer deux services : le service de base qui permet d'étendre la mémoire du client via le swap à distance ainsi que le service *PubSub* qui permet d'organiser et de contrôler l'allocation de mémoire dans l'infrastructure. Nous présentons en détail ces deux services dans les sections suivantes.

4.3.2 Classification des nœuds

Les informations concernant l'utilisation de la mémoire des nœuds sont récupérées par *asm-probe*. Il s'agit d'un thread lancé par *asm-manager* au démarrage d'ASM. Il récupère périodiquement les informations d'utilisation de mémoire à partir du répertoire */proc* qui contient des fichiers indiquant l'état du système d'exploitation.

asm-probe transmet alors ces informations à *asm-manager* qui se charge de la prise de décision concernant l'état du nœud et l'allocation de mémoire distante. Pour éviter les décisions erronées, *asm-manager* prend les décisions à partir de la moyenne des dernières valeurs transmises. Ces décisions dépendent du type de nœud (client, serveur ou indépendant) et de la charge actuelle de mémoire par rapport aux seuils de contrôle (MAX, MIN).

Si la décision prise par *asm-manager* conduit à un changement du type de nœud vers un état client ou un état serveur, *asm-manager* contacte le service *PubSub* pour effectuer les actions à faire selon le type de nœud. Nous détaillons dans la suite les détails d'implantation concernant la communication.

4.3.3 Service de PubSub

La communication entre les nœuds est basée sur le protocole TCP/IP. Nous considérons que le réseau est fiable : sans perte de message.

Pour assurer le service central de *PubSub* dans l'infrastructure, ASM s'appuie sur le système *ZeroMQ*[6]. Il s'agit d'un système open-source qui fournit un service de publication/souscription dans des infrastructures matérielles. Ce système fournit deux interfaces permettant aux nœuds de souscrire et de publier des informations aux abonnées. Il propose une version centralisée, où tous les messages passent par un nœud, et une autre distribuée où le service *PubSub* est assuré par plusieurs nœuds.

6. www.zeromq.org

Quand la charge mémoire sur le nœud augmente et dépasse le seuil MAX, *asm-manager* abonne le nœud auprès de *ZeroMQ* en tant que client. Pour cela, il lance un thread (*clientPubSub*) qui envoie sont adresse IP sur le port dédié à la suscription de *ZeroMQ*. Ce thread ouvre ensuite un port et attend les messages provenant de *ZeroMQ*.

Le client reçoit en retour de ce message une liste contenant l'état actuel des serveurs disponibles dans l'infrastructure. Cette liste comprend : l'adresse IP du serveur, la mémoire disponible et le nombre de clients qu'il héberge en ce moment. Cette liste est ensuite mise à jour par le thread *clientPubSub* chaque fois qu'il y a une nouvelle publication provenant des serveurs. Lorsque le nœud change de type et ne veut plus recevoir des notifications, il envoie le message *unsubscribe* comme suit : `unsubscribe clientIP`. *ZeroMQ* supprime alors le client en question de sa liste d'abonnées. Ainsi il ne reçoit plus rien de la part de *ZeroMQ*.

Lorsqu'un nœud devient serveur, il publie sur le port dédié à la publication sur le nœud de *ZeroMQ*. Le message envoyé est comme suit :

```
publish serverIP FreeMemory NbClients
```

ZeroMQ transmet l'information reçue à tous les clients abonnés au service. Quand le serveur devient indépendant, il envoie le même message en attribuant 0 à la taille de mémoire disponible dont il dispose. En recevant ce message, *clientPubSub* supprime le serveur de sa liste des serveurs disponibles.

4.3.4 Service de swap à distance

Dans Linux, le démon de swap *kswapd*[7] surveille en permanence l'état de mémoire pour vérifier le nombre de pages libres. Si ce dernier devient trop faible, le gestionnaire de mémoire utilise les supports de swap pour évacuer des pages de la mémoire principale vers les supports de swap. Ces supports peuvent être sous la forme d'une partition de disque ou un fichier d'échange. Pour utiliser ces supports, Linux utilise des périphériques de type block. Comme son nom l'indique, ces périphériques transmettent ou reçoivent les informations sous forme de paquets d'octets d'une taille fixe contrairement aux périphériques de type caractères. Ils permettent d'accéder aux informations d'une manière brute sans structure prédéfinie de données (par ex. fichier ou autre) ce qui les rend plus rapides.

Il existe plusieurs outils et protocoles qui peuvent être utilisés pour effectuer le swap à distance. NFS[8] et NBD[9] sont des exemples de ces outils. Ils peuvent être

7. *The Kernel Swap Daemon*
8. Network File System : un protocole permet à une machine d'accéder à des fichiers via le réseau
9. Network Block Device : un périphérique de type block dont le contenu est fourni par une machine distante

utilisés pour fournir des fichiers d'échange sur une machine distante pouvant être des supports de swap. Si ces fichiers sont montés en mémoire de la machine distante (à travers un RAM-disque [10]), ils peuvent être une forme de swap sur mémoire distante.

Nous avons fait le choix d'implanter un nouveau périphérique de type block. D'abord, parce que accéder à la mémoire en passant par NFS ou NBD traverse plusieurs couches ce qui dégrade les performances. Ensuite, parce que ces outils sont conçus en considérant une utilisation orientée disque, ils offrent des opérations qui optimisent le fonctionnement des disques, mais qui ne permettent pas de profiter des performances des mémoires. Par exemple, ils utilisent des queues pour effectuer des requêtes regroupées par secteur de disque dans l'objectif d'optimiser l'utilisation de ces derniers, ce qui peut avoir des coûts pour l'utilisation de la mémoire. De plus, ces outils permettent pas d'implanter d'autres fonctionnalités dont nous avons dans ASM comme la migration des serveurs, etc...

4.3.4.1 Client

asm-swap-device est un périphérique de type block utilisé pour fournir un support de swap permettant de bénéficier de la mémoire du serveur. Il permet d'accéder à la mémoire distante via le réseau en utilisant le protocole TCP/IP comme s'il s'agissait d'un périphérique de bloc local tel qu'un disque dur ou une partition.

asm-swap-device est implanté en tant que module noyau. Ainsi, il peut être installé et chargé dynamiquement dans un noyau en cours d'utilisation avec la commande *insmod* comme suit : `insmod asm-swap-device`. Cette opération initialise la structure de données interne d'*asm-swap-device* qui regroupe plusieurs variables. Les principales sont :

- *sock* : cette variable est le socket utilisé pour la communication avec le serveur. Pour son implantation, nous avons utilisé la librairie *Ksocket* [11] ;
- *size* : cette variable est la taille de mémoire fournie par le serveur ;
- *disk* : cette variable est une structure de données de type *gendisk*. Il s'agit d'un disque virtuel que le système utilise à travers des interfaces standards pour les opérations disques (lecture ou écriture).

Quand le module est inséré en mémoire, le périphérique est prêt à être instancié et associé à un serveur. Ceci se fait dynamiquement à la demande d'*asm-manager* quand le choix des serveurs est fait tel que nous expliquerons en détail dans le paragraphe 4.3.4.3. Les périphériques crées apparaissent dans le répertoire des périphériques matériels */dev*. Chaque périphérique est initialisé et peut être ensuite associé à un serveur.

10. un disque virtuel qui utilise une partie de la mémoire centrale en tant que mémoire de masse
11. un projet open-source qui fournit les interfaces socket de style BSD (à savoir : *socket, bind, listen, connect, accept*...) pour les développeurs du noyau. Site : ksocket.sourceforge.net

Une fois configuré, le système d'exploitation peut utiliser le périphérique comme s'il était une partition disque locale. Les opérations provenant du système d'exploitation sont traitées dans la méthode principale du périphérique (la méthode *request*). Cette procédure est appelée chaque fois que le noyau demande au périphérique de traiter des opérations de lecture ou d'écriture. Sa signature est la suivante :

```
static void asm_request (struct asm_swap_device *dev,
             unsigned long offset,
             unsigned long nbytes,
             char *buffer,
             int write)
```

Toute opération de lecture ou d'écriture effectuée sur le périphérique est traitée dans cette méthode. Les paramètres utilisés indiquent le périphérique utilisé (*asm_swap_device*), l'adresse de début de données (*offset*) et la taille de données à traiter (*nbytes*), un pointeur sur le tampon utilisé pour le transfert de données (*buffer*) et la nature de l'opération (*write*).

Cette méthode crée une structure de données contenant l'ensemble de ces paramètres et l'envoie au serveur en utilisant le socket ouvert avec lui (*sock*). Le comportement est différent selon l'opération à exécuter (écriture ou lecture) comme suit :

Opération d'écriture

1. création d'une structure de donnée à envoyer. Elle contient : l'adresse sur le périphérique, les données à écrire ainsi leur taille ;
2. envoie une requête TCP/IP au serveur ;
3. attente de l'accusé de réception de la part du serveur. Si au bout de certain délai cette confirmation n'arrive pas, on considère que le serveur est en panne. Le périphérique remonte l'information au *asm-manager* pour qu'il choisisse et configure un nouveau serveur tel que présenté dans le paragraphe 4.3.4.3.

Opération lecture

1. création d'une structure de donnée à envoyer. Elle contient l'adresse des données à lire sur le périphérique ;
2. envoie une requête TCP/IP au serveur ;
3. la méthode bloque en attendant le retour du serveur qui contient les pages demandées ;
4. une fois les données reçues, le tampon est rempli avec les données retournées par le serveur ;
5. si aucune donnée n'est reçue après un certain délai, elle renvoie une erreur.

La communication entre *asm-manager* et le *asm-swap-device* est assurée par un mécanisme d'appel système. Pour cela, *asm-manager* utilise la primitif *ioctrl* qui permet d'effectuer des opérations d'entrée/sortie spécifiques à un périphérique. Chaque appel doit fournir un paramètre spécifiant un code-requête à exécuter. Les principaux appels fournis par *asm-swap-device* sont les suivants :

- ASM_SET_SERVER : cet appel est fait lors de la configuration avec un nouveau serveur. Il est fait par asm-manager lorsqu'un accord a été conclu avec un serveur. Il prend en paramètre l'adresse IP du serveur, le port d'écoute ainsi la taille de mémoire fournie par ce dernier. *asm-swap-device* utilise ces informations pour configurer la socket *sock* ;
- ASM_SET_NEW_SERVER : cet appel est fait si le serveur a été changé suite à une opération de récupération de mémoire. Il prend en paramètre l'adresse IP de nouveau serveur ainsi le port d'écoute sur ce dernier. En recevant ces informations, *asm-swap-device* reconfigure la socket pour adresser les requêtes vers le nouveau serveur.

4.3.4.2 Serveur

asm-swap-server est utilisé au niveau d'un serveur pour fournir de la mémoire aux clients. Il s'agit d'un programme implanté en espace utilisateur pour fournir un accès distant à la mémoire. Il répond aux requêtes provenant des clients en se servant de la mémoire du nœud.

La structure d'*asm-swap-server* comporte plusieurs variables. Les principaux sont :

- *sock* : cette variable est un socket TCP utilisé pour la communication avec le client ;
- *port* : cette variable est le port d'écoute du serveur. Il reçoit les requêtes de clients sur ce port ;
- *size* : cette variable est la taille de la mémoire à fournir ;
- *data* : cette variable est une structure de données représentant la mémoire à fournir à un client. Il s'agit d'un tableau qui contient les pages mémoires du client (attribuées par ce dernier).

asm-swap-server est démarré par *asm-manager* quand un accord est conclu entre le client et le serveur. A son démarrage, il prend deux paramètres : la taille de mémoire qu'il va fournir au client, et le numéro de port d'écoute. Dès lors, il initialise la variable *data* en réservant une partie (de taille *size*) de sa mémoire et attend les requêtes des clients sur le port d'écoute.

4.3.4.3 Étendre la mémoire du client

Lorsque la mémoire de client est saturée, il fait le choix d'un serveur selon la politique utilisée telle que nous l'avons présentée dans le paragraphe 4.1.4.1. Une fois le choix fait, *asm-manager* implanté côté client contacte *asm-manager* implanté côté serveur pour conclure l'accord. Il lui envoie des informations concernent son adresse IP, la taille demandée de mémoire et le numéro du périphérique qui sera utilisé pour communiquer avec le serveur. Le message envoyé au *asm-manager* du serveur prend la forme suivant : `bookMemory clientIP size deviceID`

En recevant cette information, l'unité *asm-manager* du serveur crée un nouveau serveur de swap (*asm-swap-server*) et l'initialise tel que nous avons présenté au paragraphe précédent. Il envoie au client le numéro du port d'asm-swap-server. Le serveur enregistre dans un fichier les coordonnées du client (son adresse IP et le numéro de périphérique associé) et le numéro de port qu'il utilise pour le servir.

Quand *asm-manager* au niveau du client reçoit le retour du serveur (le numéro de port), il configure le périphérique pour qu'il communique avec le serveur choisi. Pour cela, il fait l'appel système *ASM_SET_SERVER* en fournissant l'adresse IP du serveur, le port d'écoute et la taille de mémoire fournie. Le périphérique configure son socket pour se connecter au serveur en question. Puis il fixe la taille de son disque virtuel pour qu'il soit visible au système avec la même taille de mémoire fournie par le serveur.

asm-magager notifie le système d'exploitation de l'existence d'un nouveau support de swap. Pour cela il utilise la commande (*swapon*) en fournissant le périphérique qu'il vient de configurer. Le gestionnaire de mémoire l'utilise quand il y a plus assez de mémoire disponible.

4.4 Synthèse

Dans ce chapitre, nous avons présenté la conception, l'utilisation et l'implantation de notre approche ASM. En se comparant avec les approches que nous avons présentées dans l'état de l'art, ASM présente le meilleur compromis et répond bien aux critères que nous avons définis dans le chapitre précédent (paragraphe 3.1.1).

Concernant l'intusivité, ASM n'est pas intrusif par rapport à l'architecture matérielle, aucune modification de celle-ci n'est requise pour pouvoir installer et utiliser ASM pour la gestion de mémoire dans l'infrastructure. De plus, les mécanismes de swap à distance fournis par ASM sont transparents aux applications. Celles-ci n'ont pas à être modifiées.

Les disques sont uniquement utilisés lorsqu'il n'y a plus de mémoire disponible dans l'infrastructure. La performance des clients et des serveurs est garantie grâce à la migration de swap fournie par ASM. Cette fonctionnalité garantit la performance des serveurs en privilégiant l'utilisation de leurs mémoires locales lorsqu'ils en ont besoin, tout en assurant la performance des clients.

La consolidation de la mémoire est assurée par ASM. En effet, les politiques d'allocation de mémoire permettent de contrôler le regroupement des pages distantes de plusieurs clients et d'utiliser ainsi le minimum de serveurs requis pour servir les clients. Évidemment, ces politiques doivent être couplées à un consolidateur global de haut niveau, par exemple un consolidateur de VM, qui prend en charge tous les types de ressources (CPU, mémoire, etc...). Il peut décider de concentrer les zones de swap sur quelques serveurs pour en éteindre d'autres. Nous fournissons une API permettant le contrôle d'ASM par un tel consolidateur. Ce dernier peut l'utiliser pour effectuer ces déplacements des zones de swap.

Dans le chapitre suivant, nous présentons des résultats obtenus avec notre système sur différentes applications.

Chapitre 5

Évaluation

Contents

Nous présentons dans ce chapitre les différentes expériences effectuées pour la validation et l'évaluation de nos contributions. Ce chapitre est organisé de la façon suivante : dans un premier temps, nous décrivons l'environnement matériel sur lequel nous nous appuyons pour la réalisation de nos expériences. Enfin, nous présentons les expériences réalisées qui se divisent en trois groupes :

- **les mesures de base :** elles concernent l'évaluation des bénéfices des mécanismes de swap à distance fourni par ASM ;
- **configuration statique :** dans ce cas, nous expérimentons ASM lorsque la charge mémoire est homogène au sein des nœuds. Ceci est valable notamment pour les applications scientifiques ;
- **configuration dynamique :** dans ce cas, nous expérimentons ASM lorsque la charge mémoire varie parmi les nœuds de l'infrastructure.

Pour chaque expérience réalisée, nous décrivons dans un premier temps le contexte de réalisation et le scénario de l'expérience. Ensuite, nous présentons et analysons les résultats obtenus.

5.1 Infrastructure matérielle

Les expériences présentées dans ce chapitre sont effectuées sur un cluster de type *Beowulf* composé de huit nœuds. Ils sont reliés entre eux via un switch de 1Gb/s. Les nœuds sont homogènes et disposent des caractéristiques suivantes :

- distribution Debian 6.0.7 ;
- 2 processeurs de type Intel Core 2 Duo 2.66GHz ;
- 4 Go de mémoire vive ;
- un disque dure de type Western Digital dont la vitesse est de 7200 RPM et la capacité et de 160 Go ;

5.2 Mesures de base

La première expérience que nous avons effectuée avait pour but de tester la performance des mécanismes de swap à distance fournis par le périphérique de swap d'ASM. Pour cela, nous comparons les coûts de swap sur le temps d'exécution d'une application dans deux cas :

- lors de l'utilisation d'une partition de disque local en tant que support de swap (ce qui est la configuration par défaut dans le système d'exploitation) ;
- lors de l'utilisation d'*asm-swap-device* pour faire du swap sur une mémoire distante.

Application

Nous n'avons pas trouvé de *benchmark* dédiées à la mesure de performance de périphériques de swap. Il existe des *benchmarks* qui sont faits pour mesurer les performances de la mémoire centrale en général comme le *benchmark* STREAM [41]. En expérimentant avec STREAM, nous avons constaté que son utilisation ne permet pas d'allouer une quantité de mémoire dépassant la taille de la mémoire principale. C'est pourquoi nous avons développé un benchmark (que nous appelons *StressRam*) qui permet de créer une charge synthétique de mémoire. Il peut être utilisé pour allouer une grande quantité de mémoire d'un nœud et provoquer ainsi des opérations de swap.

StressRam permet de créer deux types de profils de charge mémoire : un profil statique où il va réserver une taille fixe de mémoire et un profil dynamique où il va injecter progressivement une charge mémoire.

En lançant *StressRam* il faut préciser plusieurs paramètres, notamment :

- le type de profil de charge ;
- la taille de la mémoire à allouer ;

– le pas d'avancement et le temps d'attente (lorsqu'il s'agit d'un profil dynamique de charge).

Scénario

Dans cette expérience, nous comparons deux scénarios :
– SwapSurDisque : Linux utilise le swap sur disque local (configuration par défaut) ;
– ASM : nous installons *asm-swap-device* et nous le configurons pour qu'il utilise la mémoire d'un deuxième nœud. Nous configurons Linux pour qu'il l'utilise en tant que support de swap.

Nous nous intéressons à mesurer le temps d'exécution de *StreeRam* lorsqu'il est utilisé pour charger la mémoire et causer des opérations de swap. Pour cela, nous l'exécution plusieurs fois en changeant à chaque fois la charge mémoire (pour changer la taille du swap à chaque exécution) ce qui permet de calculer le débit moyen d'écriture sur le support de swap. L'expérience se déroule donc comme suit :

1. une première instance de *StressRam* est exécutée pour générer une charge mémoire. Cette charge doit être assez important pour occuper une grande partie de la mémoire sans pour autant causer des opérations de swap ;
2. une deuxième instance de *StressRam* est ensuite exécutée pour générer une charge mémoire équivalente à 2 Go. Nous mesurons le temps d'exécution de cette instance.
3. Nous répétons cette opération plusieurs fois en variant la charge mémoire créée par le premier *StressRam* à chaque itération et nous calculons le temps d'exécution de *StressRam* à chaque fois.

Résultat et discussion

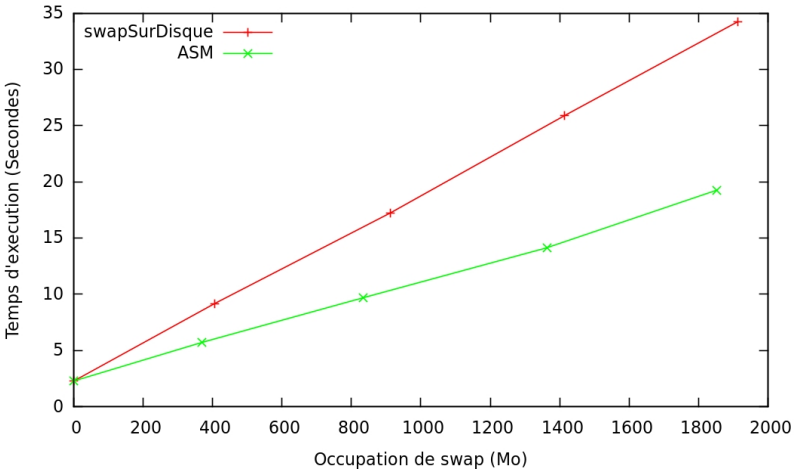

FIGURE 5.1 – Mesure de performance d'*asm-swap-device*

Les figures 5.1 et 5.2 montrent le résultat obtenu de cette expérience. Sur la figure 5.1, nous pouvons voir le temps d'exécution de *StressRam* (pour allouer 2 Go de mémoire) selon la taille de swap. Cette opération ne prend qu'environ 2 secondes lorsque la mémoire est disponible (allocation faite sans devoir utiliser le swap). Nous pouvons voir la dégradation de performance en termes de temps d'exécution lorsque la mémoire est chargée et qu'il y a des pages à évacuer de la mémoire vers le support de swap. La relation est linaire entre l'occupation du swap et le temps d'exécution de *StressRam*.

Nous pouvons constater que les mécanismes de swap assurés par ASM sont avantageux par rapport à ceux fournis par défaut sur le disque. Les opérations d'écritures sur la mémoire distante sont avantageuses pour les performances vu l'écart de vitesse entre le disque et la mémoire et malgré le réseau.

La figure 5.2 montre le débit moyen d'écriture sur le support de swap. Elle montre que lors de l'utilisation d'ASM le débit passe de 51 à 85 Mo/s ce qui représente une amélioration de performance d'environ 40%.

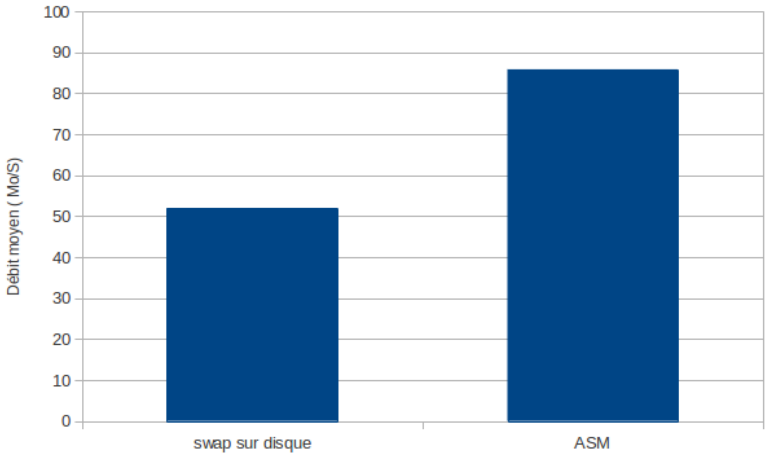

FIGURE 5.2 – Débit moyen d'écriture

5.3 Configuration statique

L'objectif de cette expérience est d'évaluer les bénéfices d'ASM lors de son utilisation pour la gestion mémoire des nœuds possédant une charge homogène de mémoire.

Ce patron d'utilisation de mémoire est valable pour une famille d'applications qui exécutent des tâches similaires et donc les nœuds sur lesquels elles s'exécutent ont une charge équivalente de mémoire. Il est valable notamment pour les applications scientifiques qui se caractérisent par une phase longue d'utilisation intensive de

mémoire. L'utilisation de mémoire dans ces applications est généralement homogène dans tous les nœuds participant au calcul.

Puisque l'application définit à priori les tâches à affecter à chaque nœud selon leurs ressources et le calcul à effectuer, nous désignons alors un ensemble de nœuds qui agissent comme serveurs de mémoire et configurons ASM pour fournir un espace initial de swap pour chaque nœud participant au calcul. ASM peut ensuite pendant l'exécution adapter l'utilisation de la mémoire selon la charge mémoire de chaque nœud.

Application

Pour effectuer cette expérience, nous avons utilisé MUMPS [2] (*MUltifrontal Massively Parallel Solver*). Il s'agit d'un solveur parallèle de systèmes d'équations linéaires (sous la forme `Ax = b` où A est une matrice creuse).

Il a été développé dans le cadre d'une collaboration entre plusieurs laboratoires de recherche (Lyon, Toulouse et Bordeaux). Il permet la résolution de grands systèmes linéaires creux pour différents types de matrices.

La résolution d'un problème linéaire avec MUMPS passe par trois étapes : les phases d'analyse, de factorisation et de résolution. Ces phases peuvent être parallélisées grâce à l'utilisation de MPI [1]. A la fin de la phase d'analyse, MUMPS donne une estimation de la mémoire requise pour effectuer le calcul.

La résolution de problème des équations linéaires consomme beaucoup de mémoire [50]. C'est pourquoi MUMPS offre une version avec un algorithme de mémoire externe (*Out-of-core*) qui permet d'utiliser le disque lorsque la mémoire ne peut pas supporter la charge produite lors de calcul. Cette version, plus complexe à développer, est plus efficace que la version par défaut lorsque la charge générée par MUMPS dépasse la taille de mémoire principale, car MUMPS a la connaissance des données utilisées pendant le calcul et peut donc placer efficacement les pages sur les disques en fonction de la localisation des processus qui en auront besoin.

Scénario

Dans cette expérience, nous comparons la performance en termes de temps d'exécution de MUMPS dans trois scénarios :

- SwapSurDisque : Linux utilise le disque local en tant que support de swap ;
- OutOfCore : dans ce cas, MUMPS est exécuté avec la version *OutOfCore*. Il n'y a donc pas d'utilisation de swap géré par Linux ;

1. Message Passing Interface : une norme définissant une bibliothèque de fonctions, utilisable avec les langages C, C++ et Fortran. Elle permet d'exploiter des ordinateurs massivement parallèles ou multiprocesseur par passage de messages.

– ASM : nous installons *asm-swap-device* et nous le configurons pour qu'il utilise la mémoire d'un nœud fixé à priori. Nous configurons Linux pour qu'il l'utilise en tant que support de swap.

Pour cette expérience nous utilisons quatre nœuds : un nœud central utilisé pour le déploiement du calcul sur les trois autres nœuds. MUMPS a été configuré pour faire, un calcul avec une grande matrice pour générer une utilisation de mémoire dépassant d'environ 3 Go la mémoire totale fournie par les trois nœuds. L'expérience dure environ 40 minutes.

Lors de l'utilisation avec ASM, nous avons utilisé le nœud de déploiement en tant que serveur qui fournit de la mémoire aux nœuds participant au calcul. La configuration des périphériques de swap a été effectuée à priori sur les trois nœuds.

Résultat et discussion

La figure 5.5 montre le résultat obtenu. Nous y trouvons le temps d'exécution totale des calculs effectués par MUMPS lors des trois scénarios présentés.

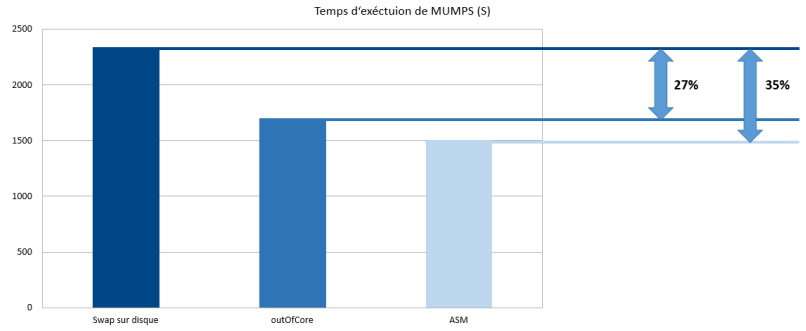

FIGURE 5.3 – Mumps

Nous pouvons constater que l'utilisation d'ASM améliore la performance en matière de temps d'exécution du calcul par MUMPS. En effet, la mémoire fournie par le nœud central a été bénéfique pour les nœuds de calcul. Les 3 Go qui sont stockés sur le disque peuvent être maintenus en mémoire lors de l'utilisation d'ASM.

La version *OutOfCore* est meilleure en termes de temps d'exécution en la comparant avec le swap sur disque géré par Linux. En effet, MUMPS place efficacement les données générées au cours du calcul entre la mémoire et le disque parce qu'il a une connaissance de ces données contrairement aux opérations de swap qui sont conçues pour être génériques.

Nous pouvons constater que le temps d'exécution diminue d'environ 35% lors de l'utilisation d'ASM par rapport au swap sur disque. Il présente un meilleur résultat que la version *OutOfCore* qui diminue déjà le temps d'exécution de 27%. ASM peut donc, sans modification de l'application, obtenir les mêmes performances (ou

même de meilleures performances) qu'une version *OutOfCore* qui est très complexe à développer.

5.4 Configuration dynamique

L'objectif de cette expérience est d'étudier le fonctionnement et le gain d'ASM lors de la gestion de mémoire dans une infrastructure à charge dynamique de mémoire. Les nœuds sont classés selon leur utilisation de mémoire en client, serveur ou indépendant. Ils s'échangent des messages par le biais de service de *PubSub* pour avoir des informations d'utilisation de mémoire dans l'infrastructure.

Pour générer une charge dynamique de mémoire qui varie d'un nœud à l'autre, nous utilisons *stressRam* que nous avons présenté dans le paragraphe 5.2. Il a été utilisé pour créer plusieurs profils de charge qui montent progressivement l'occupation de la mémoire. Le service *PubSub* est assuré par *ZeroMQ* qui est installé sur un des nœuds participant à l'expérience.

Scénario

Nous avons utilisé quatre nœuds pour réaliser cette expérience (identifiés par nœud1, nœud2...). Le système qui assure le service de *PubSub* (*ZeroMQ*) a été installé sur le nœud3. Ensuite, ASM est installé sur les quatre nœuds en suivant la procédure d'installation que nous avons décrite dans le paragraphe 4.2.2.

Une fois ASM démarré, nous créons différents profils de charge sur les nœuds en utilisant *StressRam*. Ces profils sont présentés dans la figure 5.4. Nous définissons trois phases dans cette expérience qui se déroule comme suit :

1. deux nœuds (nœud1 et nœud2) subissent une montée progressive de charge de mémoire qui dépasse la taille de la mémoire locale et qui va ainsi causer des opérations de swap. La mémoire du nœud1 reste saturée jusqu'à la fin de l'expérience. La charge de la mémoire du nœud2 dure moins longtemps puis la mémoire est libérée. Les autres nœuds (nœud3 et nœud4) restent sans charge mémoire ;
2. une fois la mémoire du nœud2 libérée, nous créons une charge mémoire sur le nœud3. Cette charge occupe une grande partie de sa mémoire sans causer d'opérations de swap. La mémoire est ensuite libérée ;
3. la mémoire du nœud1 est libérée progressivement. Une fois libérée, l'expérience est terminée.

ASM a été configuré avec les paramètres suivant :
– la politique *mostUsedServer* pour l'allocation de mémoire. Les clients choisissent donc les serveurs qui hébergent déjà d'autres clients dans la mesure du possible ;

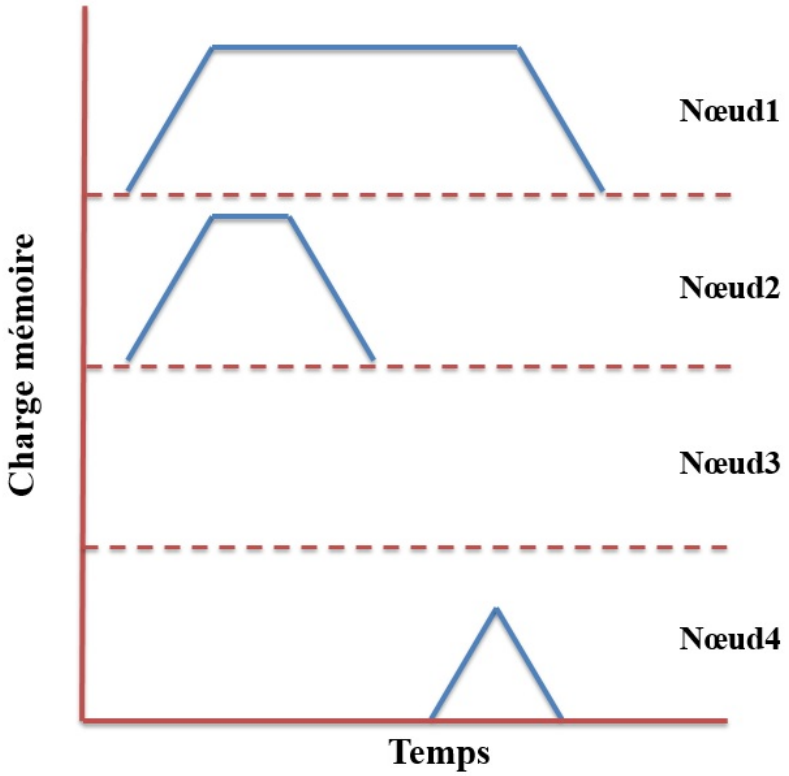

FIGURE 5.4 – Profils de charge sur les nœuds

- la politique *localMemory* pour la libération de mémoire. Dès que de la mémoire du client devient disponible, il récupère ses pages hébergées dans les autres serveurs ;
- la politique *freeWhenFinished* pour la récupération de mémoire des serveurs : le serveur supprime les pages une fois que la migration des pages est terminée ;

StressRam a été configuré pour injecter une charge progressive de mémoire aux nœuds. Il injecte 50 Mo/s pour créer une charge globale de 5.8 Go sur le nœud1 , et 4.5 Go sur le nœud2 pour la première phase. Le nœud3 est chargé à 2.5 Go pendant la deuxième phase. *StressRam* libère ensuite la mémoire qu'il a alloué de la même manière (50 Mo/s).

Pendant l'expérience, nous stockons les informations d'utilisation de la mémoire de chaque nœud pour observer le comportement d'ASM. Les résultats sont présentés dans le paragraphe suivant.

Résultat

La figure 5.5 montre la trace d'utilisation de la mémoire des quatre nœuds participant à l'expérience. Les informations qui s'y trouvent sont :

- l'occupation locale de la mémoire (courbe en rouge). Elle montre la charge locale du nœud (mémoire utilisée par les processus locaux au nœud) ;
- mémoire hébergée (courbe en vert). Elle montre la mémoire qui a été allouée aux clients (lorsque le nœud est serveur) ;
- l'occupation du swap (courbe en bleu). Elle montre la charge de swap (la taille des pages stockées sur les mémoires distantes lorsque le nœud est client) ;
- le swap alloué (courbe en violet). Elle montre la mémoire distante allouée (chunks) sur des mémoires distantes (lorsque le nœud est client).

Au début de l'expérience, tous les nœuds s'identifient en tant que serveur de mémoire au vu de leur disponibilité de mémoire. Ils publient alors la mémoire qu'il fournissent à travers *ZeroMQ*.

La charge mémoire crée par *StressRam* sur les nœuds 1 et 2 les fait passer au type indépendant puis client. Quand leur charge mémoire dépasse 80%, ils créent de nouveaux périphériques swap et les configurent pour l'utilisation de mémoire fournie par les serveurs. La politique utilisée pour l'allocation de mémoire fait qu'ils utilisent un serveur (nœud3) jusqu'à ce qu'il ne dispose plus de mémoire à fournir aux clients. Ils s'appuient ensuite sur le nœud4 pour répondre à leurs besoins de mémoire.

Nous pouvons voir cette opération sur la figure 5.5. Au niveau du nœud1, le swap alloué augmente dynamiquement chaque fois que sa charge totale de mémoire passe en dessus de 80% (les points a1, a2, ...). Ce swap alloué correspond à la mémoire hébergée des serveurs (les points c1, c2 ...au niveau du nœud3, et d1, d2 ...au niveau du nœud4). Le gestionnaire de mémoire bénéfice de ce swap pour évacuer des pages. Nous observons le même comportement au niveau du nœud2.

La charge mémoire baisse progressivement sur le nœud2. Pendant cette phase, il libère le swap alloué (selon la politique *localMemory*) à chaque fois que sa mémoire devient disponible. On observe cette baisse du swap alloué aux points b4, b5 et b6 sur le nœud2 et logiquement la baisse du swap hébergé des serveurs (les points c6 c7 au niveau du nœud3 et le point d4 au niveau du nœud4).

Dans une phase suivante, *StressRam* est démarré sur le nœud4 qui est classé serveur et héberge toujours des pages du nœud1. Suite à la montée de sa charge de mémoire locale, un événement de récupération de mémoire se déclenche. Il choisit le nœud3 en tant que nouveau serveur selon la politique utilisée (mostUsedServer) puisque nœud3 possède déjà des clients. Nœud4 déplace ainsi la mémoire vers nœud3 (les points c8 et c9 au niveau nœud3) et libère sa mémoire (les points d6 et d7 au niveau nœud4) une fois la migration est terminée. Puisque nœud3 ne dispose plus de mémoire disponible, nœud4 utilise à la fin nœud2 comme serveur (b7) et libère sa mémoire de la même façon (d8). nœud1 est averti à la fin de chaque migration de l'emplacement de ses pages.

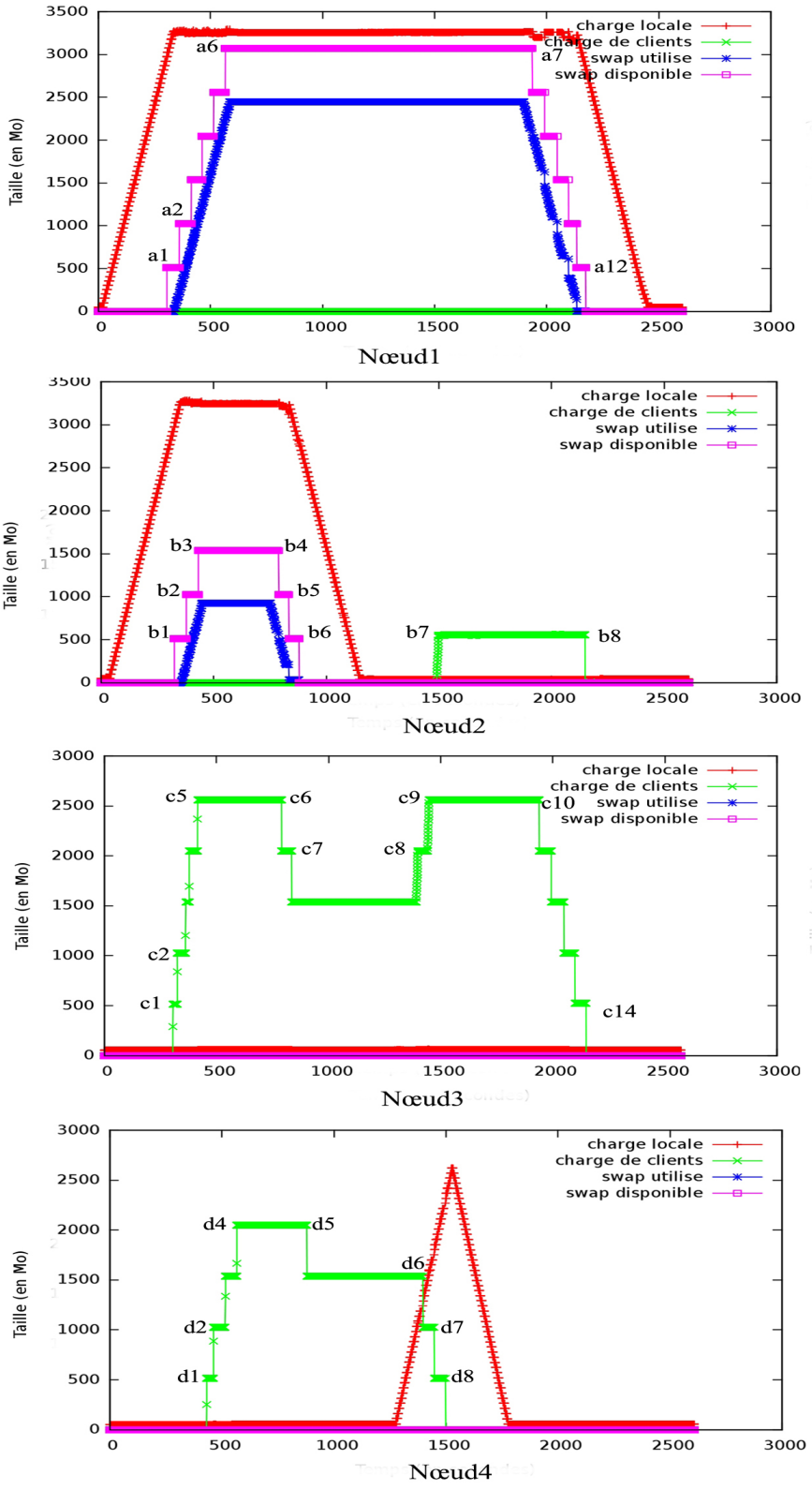

FIGURE 5.5 – Expérience avec une configuration dynamique

À la fin de l'expérience, la charge mémoire baisse progressivement sur nœud1, il libère pendant cette opération les mémoires allouées chez les nouveaux serveurs (nœud2 et nœud3).

Chapitre 6

Conclusion et perspectives

Contents

6.1 Conclusion

Les infrastructures réparties sont utilisées de plus en plus fréquemment par beaucoup d'organisations. Elles fournissent une capacité élevée de calcul et de stockage permettant de répondre à une grande charge de travail.

La gestion des ressources est un des principaux défis pour les fournisseurs de telles infrastructures. Cette opération est essentielle puisqu'elle contrôle la qualité du service fourni aux clients par l'infrastructure et elle contrôle également les coûts de fonctionnement de l'infrastructure (notamment la consommation d'énergie). La gestion de ressources vise donc à garantir la qualité de service et en même temps à gérer de façon optimale les ressources.

La gestion globale de la mémoire dans une infrastructure répartie est une problématique qui a été relativement peu explorée (par rapport à la gestion globale du processeur). Les fournisseurs s'appuient soit sur des approches locales qui se basent sur l'utilisation des disques en tant qu'extension de la mémoire locale, soit sur des approches globales qui répartissent la charge mémoire sur plusieurs machines

en s'appuyant principalement sur des mécanismes de migration de processus ou de réplication des traitements.

En réalisant un état de l'art de ces approches, nous avons constaté qu'elles présentent des inconvénients, notamment par rapport aux performances à l'exécution, à l'optimisation de la gestion des ressources ou à l'intrusivité de l'implantation dans les systèmes.

L'objectif de la thèse est la conception et l'implantation d'une approche de gestion globale de mémoire dans une infrastructure matérielle. Après l'étude des problèmes engendrés par les autres approches de gestion globale de la mémoire, nous avons pris le parti de créer ASM : un système de gestion mémoire qui permet d'éviter le gaspillage de la ressource mémoire et améliore les performances des applications tout en optimisant l'utilisation des ressources dans l'infrastructure. ASM permet à une machine, plutôt que d'utiliser son disque local en tant que support de swap, de swapper sur la mémoire distante d'une autre machine ayant de la mémoire disponible. Il permet également un équilibrage dynamique de la charge du swap entre les machines distantes hébergeant le swap.

Les résultats des évaluations effectuées avec ASM montrent une vraie amélioration de QoS des applications. Les opérations de swap sur mémoire distante gérées par ASM sont efficaces et permettent d'exploiter la mémoire disponible dans l'infrastructure et d'éviter ainsi les opérations sur les disques. Les mesures concernant les mécanismes de base d'ASM montrent une amélioration de performances d'environ 40% par rapport à une solution reposant sur un swap sur disque. Les évaluations avec l'application MUMPS montrent que le temps d'exécution d'un problème linéaire diminue d'environ 35% lors de l'utilisation d'ASM par rapport au swap sur disque. Il présente un meilleur résultat que la version *OutOfCore* qui diminue déjà le temps d'exécution de 27%. ASM peut donc, sans modification de l'application, obtenir les mêmes performances (ou même de meilleures performances) qu'une version *OutOfCore* qui est très complexe à développer.

6.2 Perspectives

Tout au long de la réalisation de cette thèse, nous avons identifié différents axes potentiels de prolongement de nos travaux. Ils peuvent être classés en deux catégories : les travaux réalisables dans un futur proche concernant le prototype ASM, et ceux réalisables dans un futur plus lointain concernant de nouvelles pistes de recherche. Nous présentons ces perspectives dans les sections suivantes.

6.2.1 Perspectives à court terme

ASM est un premier prototype pour lequel nous envisageons très prochainement deux séries de travaux : l'implantation d'un service de tolérances aux pannes et l'évaluation d'ASM dans des conditions réelles d'un centre de données. Nous étudions dans cette section les pistes de chacun de ces points.

6.2.1.1 Tolérance aux pannes

Dans une infrastructure matérielle, un nœud peut tomber en panne à tout moment. Dans la version actuelle d'ASM, nous ne prenons pas en charge les pannes qui peuvent arriver pendant l'exécution. Ceci peut arriver au niveau d'un client, d'un serveur ou du service *PubSub*. Une première amélioration d'ASM consiste donc à introduire des mécanismes de tolérance aux pannes à ces trois niveaux. Nous les détaillons dans les paragraphes suivants.

Panne au niveau du PubSub

Dans la version actuelle d'ASM, le service *PubSub* est centralisé et assuré par un seul nœud. Si ce nœud tombe en panne, les nœuds n'arrivent plus à échanger des messages pour accéder aux informations de disponibilité de la mémoire dans l'infrastructure. Ceci implique le dysfonctionnement total d'ASM puisque les nœuds utiliseront leurs disques locaux en tant que support de swap comme si ASM n'existait pas.

Pour répondre à cette situation, nous proposons d'utiliser un service *PubSub* distribué. Dans ce cas, on utilise plusieurs nœuds pour assurer le service de*PubSub*. *ZeroMQ* propose un serveur distribué que nous envisageons d'intégrer dans notre implantation.

Panne au niveau du client

ASM doit prendre en compte le cas où le client tombe en panne. Dans ce cas, il faut libérer la mémoire allouée par ce client dans différents serveurs pour qu'elle soit accessible aux autres clients.

Pour ce faire, les serveurs peuvent s'assurer que les clients qui utilisent leur mémoire existent toujours en envoyant envoyer des messages et en attendant le retour des clients. Si au bout d'un certain délai ils ne reçoivent rien, ils considèrent que le client est en panne et peuvent ainsi libérer la mémoire.

Panne au niveau du serveur

Si le nœud qui tombe en panne agit en tant que serveur, il perdra les pages de plusieurs clients. ASM doit être en mesure de récupérer les pages de ces clients, sinon ces clients tombent aussi en panne (leur swap n'étant plus accessible).

Pour répondre à cette situation, une possibilité est d'implanter un service de réplication des pages. Ainsi, quand le client fait une opération *swapout*, la page en question est envoyée à deux serveurs différents. De cette manière, lorsqu'un serveur tombe en panne, le client peut s'adresser à l'autre serveur pour récupérer ses pages. Cette solution est cependant très coûteuse en mémoire.

Une autre possibilité (si les machines possèdent des disques) est alors de réaliser une copie sur disque des pages que les clients exportent vers les serveurs. Cette copie peut être réalisée de façon asynchrone pour ne pas pénaliser les performances. Si un serveur tombe en panne, ses clients peuvent accéder à leur swap depuis leur disque local.

6.2.1.2 Évaluation en conditions réelles

Dans le chapitre 5, nous avons présenté les expériences que nous avons menées avec ASM. Nous avons utilisé l'application MUMPS (application de calcul scientifique très utilisée) pour l'évaluation de la configuration statique (simples clients et serveurs avec une charge constante). Nous avons également expérimenté avec une configuration dynamique (multiples clients et serveurs avec une variation de charge) avec un *benchmark* produisant une charge synthétique. Il serait intéressant d'évaluer cette configuration dynamique dans un domaine applicatif plus réaliste.

Nous comptons effectuer une évaluation basée sur une utilisation réelle d'un centre de données. Pour cela, nous utiliserons des données représentant des traces d'utilisation réelle d'un centre de données (*Eolas Datacenter*[1]) avec qui nous collaborons à travers un projet ANR (ctrl-green). L'idée est d'analyser ces données dans l'objectif de construire une simulation basée sur les traces[2] afin d'en déduire les performances d'ASM.

1. Site : www.businessdecision-eolas.com
2. En anglais : *trace based simulation*

6.2.2 Perspectives à moyen terme

6.2.2.1 Utilisation dans une IaaS

Nous comptons intégrer ASM dans un cloud géré comme IaaS. En général, un consolidateur de VMs est utilisé dans de telles infrastructures. Il place les VMs en fonction de leur état d'utilisation de ressources (principalement CPU et mémoire). En général, la consolidation est limitée par l'empreinte mémoire des VMs. En effet, une VM qui utilise peu de CPU aura toujours une empreinte mémoire significative, ne serait ce que parce qu'elle exécute un système d'exploitation.

Pour augementer le taux de la consolidation, les consolidateurs peuvent utiliser des techniques de *ballooning* [63]. Le *ballooning* permet de réclamer des pages mémoires à une VM en la forçant à swapper. On peut ainsi placer plus de VMs sur la même machine physique. Mais le swap disque peut pénaliser énormément les performances, alors que certaines machines (même si leur CPU est totalement utilisé) peuvent disposer de mémoire disponible.

Il devient intéressant d'intégrer un système comme ASM dans le schéma de gestion de mémoire d'un IaaS utilisant la consolidation et le *ballooning* car le coût de swap sera alors réduit et on pourra utiliser la mémoire disponible sur les machines qui en ont (celles dont les VMs utilisent plus de CPU que de mémoire).

6.2.2.2 Exploitation des machines libérées

Nous avons vu qu'ASM devrait être couplé à un consolidateur qui, lorsqu'il libère une machine (en la mettant en veille), demande à ASM de déplacer les pages hébergées par cette machine vers d'autres machines serveur restant actives (section 4.4).

Nous étudions actuellement la possibilité d'accéder à la mémoire d'une machine lorsqu'elle est en veille. Pour ce faire, nous souhaitons exploiter la technologie RDMA (*Remote Direct Memory Access*) [55] fournie notamment par les interfaces *infiniband*.

Cela permettrait de ne pas déplacer les pages hébergées lorsqu'on libère une machine, mais également d'offrir au client comme support de swap l'ensemble des mémoires des machines en veille, augmentant ainsi les possibilités de consolidation.

Bibliographie

[1] Acharya, A. and Setia, S. (1999). Availability and utility of idle memory in workstation clusters. *SIGMETRICS Perform. Eval. Rev.*, 27(1).

[2] Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., and Koster, J. (2001). A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1).

[3] Anderson, D. P. (2004). Boinc : A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*. IEEE.

[4] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). Seti@ home : an experiment in public-resource computing. *Communications of the ACM*, 45(11).

[5] Arpaci, R. H., Dusseau, A. C., Vahdat, A. M., Liu, L. T., Anderson, T. E., and Patterson, D. A. (1995). *The interaction of parallel and sequential workloads on a network of workstations*, volume 23. ACM.

[6] Baker, M., Apon, A., Buyya, R., and Jin, H. (2002). Cluster computing and applications. *Encyclopedia of Computer Science and Technology*, 45(Supplement 30).

[7] Bakshi, K. (2009). Cisco cloud computing - data center strategy, architecture, and solutions point of view white paper for u.s. public sector 1st edition. http ://www.cisco.com/web/strategy/docs/gov/CiscoCloudComputing_WP.pdf.

[8] Barak, A. and Braverman, A. (1998). Memory ushering in a scalable computing cluster. *Microprocessors and Microsystems*, 22(3).

[9] Barak, A., Guday, S., and Wheeler, R. G. (1993). *The MOSIX Distributed Operating System : Load Balancing for UNIX*. Springer-Verlag New York, Inc.

[10] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*.

[11] Becker, D. J., Sterling, T., Savarese, D., Dorband, J. E., Ranawak, U. A., and Packer, C. V. (1995). Beowulf : A parallel workstation for scientific computation. In *Proceedings, International Conference on Parallel Processing*, volume 95.

[12] Boden, N. J., Cohen, D., Felderman, R. E., Kulawik, A. E., Seitz, C. L., Seizovic, J. N., and Su, W.-K. (1995). Myrinet : A gigabit-per-second local area network. *Micro, IEEE*, 15(1).

[13] Bouchenak, S., De Palma, N., Hagimont, D., and Taton, C. (2006). Autonomic management of clustered applications. In *IEEE International Conference on Cluster Computing*.

[14] Broto, L., Hagimont, D., Stolf, P., Depalma, N., and Temate, S. (2008a). Autonomic management policy specification in tune. In *Proceedings of the 2008 ACM symposium on Applied computing*.

[15] Broto, L., Hagimont, D., Stolf, P., Depalma, N., and Temate, S. (2008b). Autonomic management policy specification in tune. In *Proceedings of the 2008 ACM symposium on Applied computing*.

[16] Buytaert, K. et al. (2002). The openmosix howto. *The Linux Documentation Project*.

[17] Cappello, F., Caron, E., Dayde, M., Desprez, F., Jegou, Y., Primet, P., Jeannot, E., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Quetier, B., and Richard, O. (2005). Grid'5000 : A large scale and highly reconfigurable grid experimental testbed. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*.

[18] Caron, E. and Desprez, F. (2006). Diet : A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3).

[19] Cecchet, E. (2002). Memory mapped networks : a new deal for distributed shared memories ? the scifs experience. In *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*.

[20] Chen, G., Malkowski, K., Kandemir, M., and Raghavan, P. (2005). Reducing power with performance constraints for parallel sparse applications. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*.

[21] Chen, H., Luo, Y., Wang, X., Zhang, B., Sun, Y., and Wang, Z. (2008). A transparent remote paging model for virtual machines. In *International Workshop on Virtualization Technology (IWVT'08)*.

[22] Cignoni, P., Montani, C., Puppo, E., and Scopigno, R. (1996). Optimal isosurface extraction from irregular volume data (addendum).

[23] Dean, J. (2009). Designs, lessons and advice from building large distributed systems. *Keynote from LADIS*.

[24] (EPA), U. E. P. A. (August 9, 2007). Report to congress on server and data center energy efficiency.

[25] Farias, R. and Silva, C. T. (2001). Out-of-core rendering of large, unstructured grids. *IEEE Comput. Graph. Appl.*, 21(4).

[26] Feeley, M. J., Morgan, W. E., Pighin, E., Karlin, A. R., Levy, H. M., and Thekkath, C. A. (1995). Implementing global memory management in a workstation cluster. In *ACM SIGOPS Operating Systems Review*, volume 29.

[27] Foster, I. (2003). The grid : A new infrastructure for 21st century science. *Grid Computing : Making the Global Infrastructure a Reality.*

[28] Gadafi, A., Broto, L., Sayah, A., Hagimont, D., and Depalma, N. (2010). Autonomic energy management in a replicated server system. In *Proceedings of the 2010 Sixth International Conference on Autonomic and Autonomous Systems.* IEEE Computer Society.

[29] Gadafi, A., Hagimont, D., Broto, L., Sharrock, R., and De Palma, N. (2011). Energy-QoS Tradeoffs in J2EE Hosting Centers. *International Journal of Autonomic Computing, IJAC.*

[30] Gunaratne, C., Christensen, K., Nordman, B., and Suen, S. (2008). Reducing the energy consumption of ethernet with adaptive link rate (alr). *Computers, IEEE Transactions on*, 57(4).

[31] Hagimont, D., Broto, L., Gadafi, A., and Depalma, N. (2012). Experience with Autonomic Energy Management Policies for JavaEE Clusters. In *Handbook of Energy-Aware and Green Computing.* Chapman & Hall, CRC Press.

[32] Hermenier, F., Lorca, X., Menaud, J.-M., Muller, G., and Lawall, J. (2009). Entropy : a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments.*

[33] Kamga, C. M., Tran, G. S., and Broto, L. (2012). Extended scheduler for efficient frequency scaling in virtualized systems. *ACM SIGOPS Operating Systems Review*, 46(2).

[34] Keir, C. C., Clark, C., Fraser, K., H, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI.*

[35] Kilburn, T., Edwards, D. B. G., Lanigan, M. J., and Sumner, F. H. (1962). One-level storage system. *Electronic Computers, IRE Transactions on*, EC-11(2).

[36] Li, K. (1988). Ivy : A shared virtual memory system for parallel computing. In *ICPP (2).*

[37] Liang, S., Noronha, R., and Panda, D. K. (2005). Swapping to remote memory over infiniband : An approach using a high performance network block device. In *Cluster Computing. IEEE International.*

[38] Lottiaux, R., Gallard, P., Vallée, G., Morin, C., and Boissinot, B. (2005). Openmosix, openssi and kerrighed : a comparative study. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2.

[39] Lowekamp, B., Tierney, B., Cottrell, L., Hughes-Jones, R., Kielmann, T., and Swany, M. (2004). A hierarchy of network performance characteristics for grid applications and services. *Proposed Recommendation GFD-RP*, 23.

[40] Markatos, E., Markatos, E. P., Dramitinos, G., and Dramitinos, G. (1996). Implementation of a reliable remote memory pager. In *In USENIX Annual Technical Conference*, pages 177–190.

[41] McCalpin, J. D. (1995). A survey of memory bandwidth and machine balance in current high performance computers. *IEEE TCCA Newsletter*.

[42] Midorikawa, H., Kurokawa, M., Himeno, R., and Sato, M. (29 2008-Oct. 1). Dlm : A distributed large memory system using remote memory swapping over cluster nodes. In *Cluster Computing, 2008 IEEE International Conference on*.

[43] Minas, L. and Ellison, B. (2009). *Energy Efficiency for Information Technology : How to Reduce Power Consumption in Servers and Data Centers*. Intel Press.

[44] Moore, R., Baru, C., Marciano, R., Rajasekar, A., and Wan, M. (1999). Data-intensive computing. *The Grid : Blueprint for a New Computing Infrastructure, Morgan Kaufmann*.

[45] Morin, C., Lottiaux, R., Vallée, G., Gallard, P., Utard, G., Badrinath, R., and Rilling, L. (2003). Kerrighed : a single system image cluster operating system for high performance computing. In *Euro-Par 2003 Parallel Processing*.

[46] Newhall, T., Finney, S., Ganchev, K., and Spiegel, M. (2003). Nswap : A network swapping module for linux clusters. In *Euro-Par 2003 Parallel Processing*.

[47] Padala, P., Zhu, X., Wang, Z., Singhal, S., Shin, K. G., et al. (2007). Performance evaluation of virtualization technologies for server consolidation. *HP Laboratories Technical Report*.

[48] Pfister, G. F. (2001). An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O*, 42.

[49] Rabaey, J. M., Chandrakasan, A. P., and Nikolic, B. (2002). *Digital integrated circuits*, volume 2. Prentice hall Englewood Cliffs.

[50] Raju, M. P. (2009). Parallel computation of finite element navier-stokes codes using mumps solver. *arXiv preprint arXiv :0910.1845*.

[51] Rambus (1999). Rambus,rdram. http :Hwww.rambus.com.

[52] Ramesh, B., Ribbens, C., and Varadarajan, S. (2011). Is it time to rethink distributed shared memory systems ? In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*.

[53] Ranganathan, K. and Foster, I. (2001). Identifying dynamic replication strategies for a high-performance data grid. *Grid Computing—GRID 2001*.

[54] Rehr, M. and Vinter, B. (2010). The user-level remote swap library. In *Proceedings of the 2010 IEEE 12th International Conference on High Performance Computing and Communications*.

[55] Riley, D. D. (2005). System and method for remote direct memory access over a network switch fabric.

[56] Samih, A., Wang, R., Maciocco, C., Tai, T.-Y. C., and Solihin, Y. (2011). A collaborative memory system for high-performance and cost-effective clustered architectures. In *Proceedings of the 1st Workshop on Architectures and Systems for Big Data*.

[57] Sharma, R., Stearns, B., and Ng, T. (2001). *J2EE Connector Architecture and Enterprise Application Integration*. Addison-Wesley Professional.

[58] Shen, H.-W., Chiang, L.-J., and Ma, K.-L. (1999). A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proceedings of the conference on Visualization '99 : celebrating ten years*. IEEE Computer Society Press.

[59] Srikantaiah, S., Kansal, A., and Zhao, F. (2008). Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*.

[60] Sterling, T. (2001). *Beowulf cluster computing with Linux*. MIT Press.

[61] Verma, A., Dasgupta, G., Nayak, T. K., De, P., and Kothari, R. (2009). Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 conference on USENIX Annual technical conference*.

[62] Vitter, J. S. (2001). External memory algorithms and data structures : Dealing with massive data. *ACM Computing Surveys*, 33.

[63] Waldspurger, C. A. (2002). Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36(SI).

[64] Yokokawa, M., Shoji, F., Uno, A., Kurokawa, M., and Watanabe, T. (2011). The k computer : Japanese next-generation supercomputer development project. In *Low Power Electronics and Design (ISLPED) International Symposium*.

Liste des tableaux

Table des figures