Fabien Hermenier

18 avril 2006

Evolution dynamique des grilles informatiques : Application à la gestion d'énergie

- Etude bibliographique -

Laboratoire : EMN-INRIA/LINA
Equipe : OBASCO

Encadrant : Jean-Marc Menaud

Master Recherche Mention Informatique, Spécialité Architectures Logicielles Distribuées

Table des matières

Chapitre 1

Introduction

Cette étude bibliographique tend à dresser un état de l'art sur les grilles informatiques et notamment de mettre en avant les problèmes de reconfiguration dynamique de l'architecture qui se posent. Nous ferons dans un premier chapitre, un aperçu de l'architecture, du domaine d'utilisation et des problèmes des grilles. Puis nous expliquerons dans une seconde partie en quoi la mise en plaçe d'un service d'économie d'énergie dans une grille nécessite la reconfiguration de son architecture. Enfin, dans une troisième partie, nous ferons un état de l'art concernant le domaine de la virtualisation et les services que cette technologie peut apporter.

Chapitre 2

Les grilles informatiques

La recherche de la plus grande puissance de calcul a toujours été un des premiers objectifs de l'informatique. Le domaine scientifique est de plus en plus gourmand en calcul et il est nécessaire de repenser régulièrement les architectures de types HPC[1] afin s'adapter aux besoins.

Les premiers systèmes orientés vers la puissance de calcul étaient les machines parallèles multiprocesseurs. Ces machines étaient très performante et les applications relativement faciles à développer. Cependant, l'architecture matérielle était peu ou pas évolutive et le coût d'acquisition de ce type de machine augmentait d'une manière exponentielle par rapport à la puissance de calcul fourni.

Dans le milieu des années 90, l'architecture de type cluster permit de faire évoluer les choses en proposant d'utiliser plusieurs machines standards et donc à faible coût d'achat et d'entretien. Ce type d'architecture est beaucoup plus évolutif et l'ajout de nouveaux noeuds permettait d'augmenter sensiblement les performances. Le développement d'applications est par contre plus délicat et il est nécessaire de prendre en compte la distribution au niveau algorithmique si l'on souhaite utiliser l'architecture au maximum de ses capacités. Ce type d'architecture a tout de même montré ses limites et l'interconnexion de cluster a commencé à faire son apparition.

Nous verrons dans une première partie une définition générale des grilles informatiques et les différents types de grilles envisageables, puis nous décrirons dans une seconde partie leurs architectures. Nous terminerons ce chapitre en évoquant les problèmes qui caractérisent les grilles informatiques,

[1] High Performance Computing

notamment celui de l'évolution dynamique de son architecture.

2.1 Présentation générale

Le concept de grille informatique a été introduit par I. Foster et Kesselman[28] comme une infrastructure massivement distribuée pour le calcul scientifique. Cette architecture devait permettre de centraliser différentes ressources distribuées géographiquement et reliées entre elles par des réseaux haut débit. Le principe des clusters a donc été repris mais élargi. Les différents composants de la grille sont variés ; on retrouve des clusters, des supercalculateurs ou même des stations de travail permettant de répondre encore une fois à un besoin de calcul énorme ou de manipulation de grands volumes de données. Une grille devient donc une organisation virtuelle, regroupant plusieurs acteurs mettant en commun une partie de leurs ressources.

On peut à l'heure actuelle définir trois types de grilles informatiques.

- Le *virtual Supercomputing*, consistant à associer plusieurs super-calculateurs entre eux répartis géographiquement, offrant la vision d'un supercalculateur virtuel
- *L' Internet Computing* combinant un grand nombre de machines (plusieurs dizaines de milliers), regroupées entre elles par le réseau internet. Ces machines sont mises à disposition de la grille lorsque leur taux d'utilisation devient faible. Cette architecture permet d'obtenir une puissance de calcul très importante mais également très variable. Le projet Seti@Home a pu ainsi obtenir une puissance de calcul disponible d'environ 62 Teraflops/s, soit quasiment le double des performances atteignables par Earth Simulator, le cluster le plus puissant du monde.
- Le *Meta computing* consiste à associer plusieurs machines disposant d'applications différentes. L'ensemble est alors vu comme un ordinateur virtuel unique proposant un vaste choix d'applications et de services. Ce système permet d'obtenir des services à la demande puisque tout est fournit par la grille : systèmes, applications et gestion des données.

2.2 Architecture des grilles informatiques

De par leurs tailles, les grilles de calculs sont amenées à avoir une structure très hétérogène et ce, à plusieurs niveaux (matériel, système et applicatif). Afin de faciliter l'interopérabilité entre les différents systèmes, l'architecture des grilles a été construite en respectant un certain nombre de protocoles standards, notamment les protocoles relatifs aux services web. Deux standards ont ainsi été définis OGSA[2] et WSRF[3]. Ian Foster et al.[10] ont décrit l'architecture d'une grille comme une accumulation de cinq couches que nous détaillerons dans les paragraphes suivant. La figure 2.1 réprésente cette architecture.

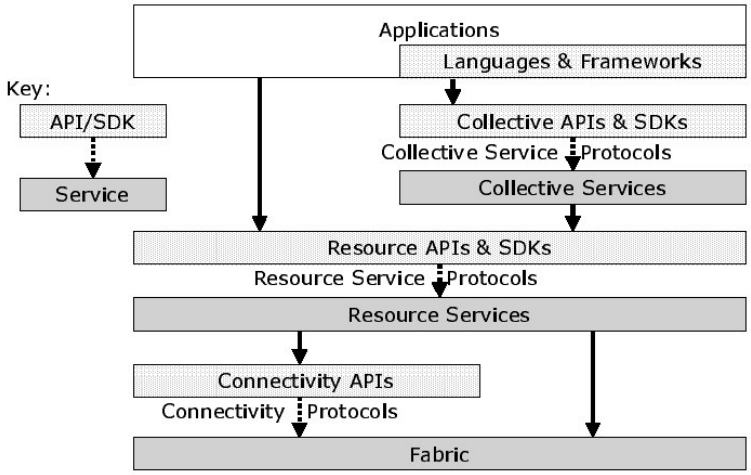

FIG. 2.1 – Architecture en couche d'une grille informatique

La couche *fabric* est la couche la plus basse de la pile. Cette couche est responsable de l'adaptation des ressources que l'organisation locale a choisi de mettre à disposition. Une ressource dans une grille est une entité pouvant fournir un service. Sa nature peut être variable : applications, données, mais aussi serveurs de fichiers, bases de données, ou super-calculateurs. L'adaptation des ressources peut être une tâche complexe, puisqu'il est nécessaire de trouver une solution permettant de satisfaire à la fois l'organisation locale et l'organisation virtuelle. Il est également nécessaire de masquer les opérations

[2]Open Grid Service Architecture
[3]WorkSpace Ressource Framework

internes qui permettent à l'organisation locale d'accéder à cette ressource. La couche *fabric* contient également les applications permettant de décrire les ressources mises à disposition. Cette description doit à la fois décrire les services rendus par la ressource, mais également ses dépendances et son état.

La couche *Resource* définit les protocoles permettant la gestion des ressources de bout en bout (initialisation, négociation, supervision, ...) en normalisant entre autre l'accès à la ressource locale. Dans un environnement hétérogène, il est en effet nécessaire de pouvoir s'adapter aux moyens d'accès aux ressources mis en place par la couche d'adaptation (différents protocoles de transports, de nommage ou de routage).

La couche *collective* contient l'ensemble des services permettant d'obtenir une vue globale de l'organisation virtuelle. Cette couche contient alors un ensemble de protocoles et d'applications permettant de découvrir les ressources mise à disposition de la grille. Les applications permettant d'augmenter la qualité de service de la grille se situent également sur cette couche (service de réplication de données par exemple). Le service d'authentification globale permettant aux utilisateurs d'être identifiés sur la grille agit également sur cette couche (bien que ce service est disponible localement sur chaque noeud de la grille). La plupart des informations sont disponibles sous la forme d'annuaires ou de catalogues. La plupart des applications de cette couche peuvent être développées sous la forme de service classique (la communication s'effectue alors par protocoles), ou sous la forme d'une API. Cette approche permet d'éviter d'avoir recours à un service centralisé et augmente la tolérance aux pannes, mais introduit une certaine lourdeur dans le développement des applications.

Enfin, la couche *applications* permet l'accès à la grille. Les applications de cette couche utilise des API dédiées au développement d'applications pour grilles permettant entre autre de réserver les ressources disponibles, d'authentifier l'utilisateur sur tous les noeuds qui seront mis à disposition et de lancer les différentes tâches.

2.3 Les problèmes des grilles informatiques

La gestion de l'évolution dans les grilles est encore problématique. Si la gestion des évolutions logicielles est envisageable par différentes techniques comme le tissage d'aspects dynamique, la mise à jour de la machine elle même nécessite un arrêt de celle-ci et donc un arrêt du service. Cet arrêt

peut mettre à mal le fonctionnement de la grille entière si ce service est critique (catalogue de ressource par exemple). Des solutions logicielles de type *heartbeat*[4] ont pu être adaptées aux grilles de calcul par Paul Stelling et al.[27] mais ces solutions ne sont pas transparentes (elles nécessitent une prise en compte aux niveaux des applications) et allourdissent donc l'organisation de la grille.

Si l'ajout de ressources est relativement aisé grace aux services de découverte, il est encore délicat de configurer les noeuds à insérer dans la grille. Chaque noeud doit être configuré pour adapter les ressources mises à la disposition de la grille. Cette adaptation se fait le plus souvent au cas par cas. De même, la reconfiguration de l'architecture de la grille est fastidieuse. Si des applications doivent être amenées à changer de noeuds pour des raisons d'optimisations par exemple, il sera nécessaire, si l'architecture matérielle et/ou système est différente, de réadadapter l'application. Edmund Smith et Paul Anderson[26] ont proposé une solution permettant la reconfiguration dynamique de la couche *fabric* mais cette solution, bien qu'ayant un déploiement pouvant être dynamique, nécessite la création de squelettes de configuration pour chaque type d'architecture.

[4]Prise de pouls

Chapitre 3

L'économie d'énergie

3.1 Enjeux

Depuis ses débuts, l'informatique s'est tournée principalement vers l'augmentation des performances des systèmes, la loi de Moore[18], annoncant que les performances des systèmes doubleraient tous les dix huit mois environ continue ainsi d'être applicable plus de quarantes années après son énonciation. Cette augmentation des performances s'est réalisée entre autre par l'augmentation du nombre de transistors des processeurs et donc par une augmentation de leur consommation électrique. Dans la majeure partie des cas, l'augmentation des besoins des utilisateurs n'a pas été proportionelle à celle des performances. Une étude de Omnigroup Consulting estime ainsi que les ordinateurs sont actuellement utilisés à 47% de leur capacités maximales. La situation peut alors être résumée de la sorte : les ordinateurs sont de plus en plus gourmands en énergie afin de nous fournir une capacité de calcul très importante mais qui n'est dans la majorité des cas pas utilisée.

Le secteur de l'économie d'énergie contrairement à celui de la haute performance n'a pas été énormément mis en avant. Ce domaine étant encore majoritairement réservé au secteur de l'informatique mobile. En effet, le besoin d'économie est important puisque le gain est direct et visible : les batteries durent plus longtemps. Ce besoin n'existe pas réellement dans l'informatique "classique" puisque les machines sont reliées en permanence au réseau électrique.

Le besoin d'économie est pourtant apparu dans le domaine des systèmes massivement distribués tel que les clusters ou les grilles informatiques. Dans

ce domaine, le nombre de machines peut être très important (l'ensemble de l'infrastructure de Google compte environ 15000 CPUs) et des coûts annexes à la consommation directe entrent en jeux. En effet, les composants tels que les microprocesseurs dégagent énormément de chaleur, et ne peuvent fonctionner sans un système de climatisation performant et coûteux. L'équation d'Arrenhius appliquée à la micro-électronique montre par exemple que chaque augmentation de 10°C de la température de fonctionnement d'un composant fait doublier son taux d'échec, réduisant ainsi sa durée de vie. Dans certains cas extrêmes, les coûts de fonctionnement et d'entretien annuels d'une grille de calcul sont quasiment égaux au coût d'achat de cette grille. Il devient donc nécessaire de trouver des techniques permettant de réduire la consommation énergétique de ce genre de systèmes, afin de faire baisser aussi bien le coût de fonctionnement de ceux-ci, mais aussi le coût d'entretien.

3.2 Principes et problématiques de l'économie d'énergie

Il a été expliqué dans la section précédente que les besoins des utilisateurs sont souvent inférieurs aux capacités des machines qu'ils utilisent. Il peut donc être envisagé de brider l'alimentation de certains des composants, abaissant les performances générales des machines, sans pour autant que l'utilisateur ne soit pénalisé (un seuil de tolérance est envisageable).

Nous verrons donc dans une première partie les différentes zones où une économie d'énergie est envisageable puis les problèmes que l'application de ces restrictions peuvent poser.

3.2.1 Les différents points d'action sur un système

La figure 3.1 permet de mettre en avant les composants consommateur d'énergie et nous permet ainsi d'en déduire les zones où l'économie d'énergie peut être la plus significative.

Le composant le plus gourmand en énergie est le microprocesseur. Il consomme à lui seul plus d'un tiers de l'énergie du système. Les premières solutions consistaient à utiliser des processeurs peu gourmands en énergie, dégageant moins de chaleur, mais aussi moins performants. Les solutions actuelles tendent vers des processeurs dont les performances sont adaptables,

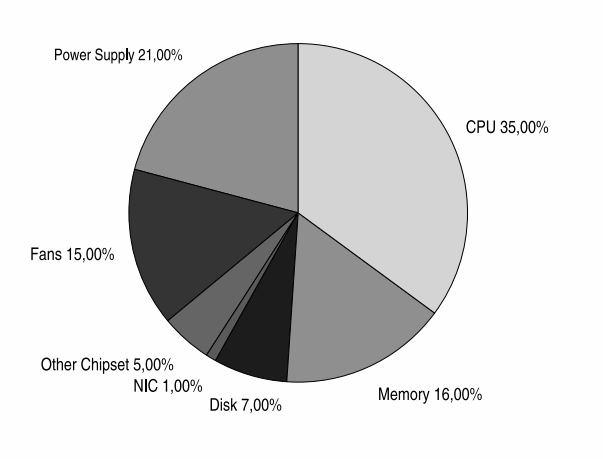

FIG. 3.1 – Répartition de la consommation énergétique pour un système isolé consommant 59 Watts

en utilisant du DVFS[1]. Cette approche consiste à proposer plusieurs niveaux d'exécution pour le processeur. A chaque niveau correspond un voltage et une fréquence de fonctionnement (voir figure 3.2) et donc une consommation électrique et des performances différentes. Le choix de ce niveau d'exécution se faisant en fonction de sa charge. Il peut être remarqué que cette économie d'énergie n'est pas directement proportionnelle à son niveau d'exécution : Les gains d'énergies sont relativements faibles face à la perte de puissance dans les niveaux les plus bas.

L'alimentation consomme elle aussi une quantité d'énergie non négligeable. Elle doit en effet alimenter divers composants qui lui sont propre comme les condensateurs ou les ventilateurs. Une partie de l'énergie est également convertie sous forme de chaleur qu'il est nécessaire de dissiper. La consommation d'une alimentation est par contre variable et dépend du besoin en énergie des différents composants de la machine (disque dur, ventilateurs, mémoire, ...). Une économie indirecte peut donc être faite en baissant la consommation des composants à alimenter. Le seul moyen de réduire directement la consommation électrique d'une alimentation étant la mise hors tension de celle-ci (et donc du système entier).

[1]Dynamic Voltage and Frequence Scaling - Adaptation dynamique du voltage et de la fréquence

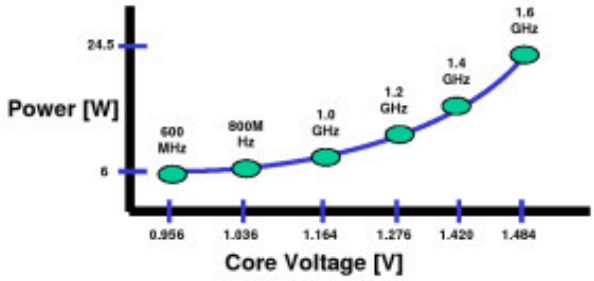

FIG. 3.2 – Variation de la consommation d'un CPU Intel Pentium M 1.6GHz à différents niveaux[12]

Modèle	Idle State	Non-idle State	Stand-by State
Seagate ST3500841NS	7.4	13.0	0.8
Western Digital WD1500AHFD	7.9	8.4	1.8
Samsung HD400LJ	8.4	8.0	0.7
Maxtor DiamondMax 11	8.1	13.6	2.2

TAB. 3.1 – Consommation électrique en Watts de disques dur, selon leur mode de fonctionnement (données constructeurs[7, 25, 24, 17])

Les disques durs consomment également une bonne proportion de l'énergie totale, cette consommation est, de plus, succeptible de varier en fonction de son utilisation. Les fabriquants proposent aujourd'hui des modes de fonctionnements permettant de mettre le disque dur en mode veille lors d'une faible activité (voir tableau 3.1). Les données à écrire sont conservées dans un cache, puis écrites une fois une quantité seuil dépassée. Il est également possible d'arrêter le moteur faisant tourner les plateaux (*spin down*).

Il est également possible d'agir sur la consommation de la mémoire. Les barrettes de types Rambus permettent par exemple de fonctionner à différents niveaux, tout comme les processeurs. La consommation de la barette diminue dans ce cas, proportionnellement à sa fréquence et son temps d'accès.

Le graphique nous montre enfin une consommation de la carte réseau relativement faible. Cela est dû au fait que l'application test n'utilise pas de fonctionnalitées réseaux pour la génération des résultats. Les cartes réseaux haut-débit (gigabit) et les commutateurs peuvent également consommer une

quantité d'énergie non négligeable. L'approche DLS[2] proposée par E.Kim et al.[15] permet de couper le lien entre la carte réseau et les commutateurs en cas de non utilisation de celle-ci. L'économie se fait ainsi au niveau de la carte réseau, mais également au niveau du commutateur (la consommation étant proportionnelle au nombres de liens actifs). Ce gain d'énergie sur les commutateurs peut-être très conséquent dans les grandes infrastructures distribuées comme les clusters, où les commutateurs ou routeurs peuvent consommer plusieurs kilowatts et dégager une chaleur importante.

La norme ACPI[3] est une norme définissant un ensemble de spécifications permettant le dialogue entre certains composants (eux-même répondant à la norme ACPI) et le système d'exploitation. La première version de cette norme a été mise au point par un ensemble d'industriel (tel que Microsost, Intel ou encore Hewlett-Packard) en 1996 et avait pour vocation de remplacer l'ancienne norme APM[4]. ACPI permet entre autre d'obtenir un ensemble d'informations sur l'état des composants compatibles tel que sa température ou son mode de fonctionnement électrique (mode veille ou normal), mais également d'agir directement sur leur mode de fonctionnement ou de consommation et cela sans redémarrage ou arrêt de service.

3.2.2 Les problèmes que posent l'économie d'énergie

Nous avons vu dans la section précédente les différents points où il est utile d'agir pour économiser de l'énergie. On ne peut cependant appliquer les stratégies sans réflexion préalable et sans connaissance de l'état du système. Il est donc souvent nécessaire de coupler le système de décision de la stratégie avec un système de monitoring qui permettra de suivre l'évolution des différents systèmes. Cependant, la quantité de données à recueillir et à analyser peut être conséquente et au final pénaliser le système. Si les données sont sauvegardées pour réaliser des hypothèses sur une longue période par exemple, il sera impossible de couper l'alimentation des disques durs. Mark Grechanik et al.[11] proposent ainsi une approche par tissage d'aspects permettant de monitorer facilement et sans grande consommation de ressources les grilles de calculs. L'aspect pouvant être tissé ou détissé dynamiquement, en fonction des besoins.

[2]Dynamic Link Shutdown
[3]Advanced Component Power Interface
[4]Advanced Power Managment

Il est également délicat de savoir quand appliquer une stratégie d'économie. Les hypothèses établient grâce aux systèmes de monitoring ne sont en effet pas toujours valable. L'activité des machines n'est forcément régulière et la mise en place des politiques peut prendre du temps et/ou des ressources. S'il est nécessaire d'arrêter ou de redémarrer une machine par exemple, son temps d'initialisation tout comme sa consommation d'énergie durant cette étape doit être prit en compte. Une stratégie pour être viable doit donc être appliquer pendant une durée minimale.

Le troisième problème vient de la dégradation possible des performances. Il est alors intéresssant de définir un seuil minimal permettant de définir si une stratégie d'économie d'énergie est viable ou pas. Ce seuil est variable et dépend entre autre des besoins des utilisateurs. Dans le cadre d'un cluster de serveur web par exemple, si une stratégie permet d'économiser 30% d'énergie, au détriment d'une latence supplémentaire de 10ms, l'économie sera considérée comme rentable. Par contre, si dans le cadre d'une grille de calcul, une économie d'énergie de 10% se fait au détriment d'un rallongement du temps de calcul identique, elle ne sera pas valable.

3.3 Les différentes approches pour l'économie d'énergie

Nous décrirons dans cette section, les principales méthodes utilisées pour réaliser des économies d'énergie, leur domaines d'application et les résultats envisageables. L'approche par optimisation du code a été volontairement mise de côté.

3.3.1 IVS - Independant Voltage Scaling

IVS est la stratégie la plus simple à mettre en oeuvre. Elle consiste à faire du DVFS sur les processeurs de chaque machine. Chaque machine analyse son état et sa charge CPU et choisit alors le niveau de fonctionnement approprié. Cette méthode a l'avantage d'être totalement indépendante de l'environnement et peut être réalisée entièrement au niveau matériel, par le processeur lui-même (c'est le cas des processeurs Crusoe), la mise en place de cette technologie ne nécessitant que des processeurs proposant ce type de service(technologie SpeedStep de Intel, ou PowerNow de AMD entre autre) . Elnozahy et al.[9] annoncent dans leur évaluation un gain d'énergie d'environ

24,5%, ce gain pouvant cependant varier de 4 ou 5 points, en fonction du type d'application.

Cette approche est également valable lorsque l'architecture ne permet pas de faire de CVS (voir section suivante), ou lorsque l'on souhaite agir plus localement (sur un noeud précis). Chen et al[5] proposent ainsi d'utiliser une approche de ce type pour réduire la consommation d'énergie des machines n'appartenant pas au "chemin critique" d'un calcul scientifique. Le chemin critique étant considéré comme l'ensemble des machines dont une baisse des performances rallongerait le temps de calcul. L'économie d'énergie est dans ce cas transparente.

3.3.2 CVS - Coordinated Voltage Scaling

CVS consiste à faire du DVFS de façon coordonnée sur plusieurs machines. Chaque machine collabore avec les autres afin de connaître la charge moyenne globale pour se mettre à son niveau. Périodiquement, chaque machine envoie à un moniteur centrale son état, celui-ci récolte les informations et en déduit la charge moyenne. Il diffuse ensuite cette information à toute les machines qui n'ont plus qu'à adapter leur niveau. Cette stratégie est plus délicate à mettre en place que IVS puisque elle nécessite d'une part, des processeurs dont le mode d'exécution est configurable de manière logiciel (tout comme IVS) ainsi qu'un service client/serveur chargé d'envoyer les requêtes de mise à niveau et de récolte des données.

Cette stratégie s'adapte très bien aux clusters où un système de répartition des charges assure une distribution uniformes. Dans ce genre de configuration, le gain d'énergie estimable est d'environ 25.5%. Ce gain reste supérieur au gain obtenus par IVS, mais la faible différence vient principalement de la compléxité supérieur de l'implémentation de la stratégie. Ce système est par contre moins efficace dans les grilles de calculs ou dans des systèmes où la charge varie énormément d'une machine à l'autre. Dans ce cas, certaines machines très chargées auront tendance à se mettre en mode dégradé afin de se mettre au niveau global de la grille, faisant chuter les performances.

3.3.3 VOVO - Vary On, Vary off

Cette technique, mise au point par Pinheiro *et al.*[19] et Chase *et al.*[4] consiste à adapter le nombre de noeuds disponibles pour un cluster en fonction de sa charge et de ses performances. Cette technique utilise aussi bien les

principes de la répartition que la concentration des charges. Périodiquement, un algorithme situé sur la machine effectuant la répartition estime s'il est nécessaire ou non d'aggrandir ou de réduire le nombre de noeuds actifs dans le cluster. S'il est décidé d'agrandir le nombre de noeuds, des machines sont mises en marche et rejoignent l'ensemble des noeuds grâce au *Wake On Lan*. Si le nombre de machine doit être réduit, des machines sont ciblées et doivent transmettre leurs tâches avant de s'éteindre. Dans ce cas, celles-ci sont envoyées sur les machines les moins chargées. Un seuil de tolérance est défini afin de spécifier un ratio performance/économie d'énergie correct. Il est ainsi envisageable de surcharger légèrement une machine, si cela permet d'en éteindre plusieurs.

Cette stratégie permet une grosse économie puisque les implémentations des deux auteurs ont permis une économie avoisinants les 31,5 %. Ce gain est cependant très variable, puisque entièrement dépendant de la variation de la charge du cluster en fonction du temps (L'article de Pinheiro et al.[19] constate même un gain de l'ordre de 86% dans certains cas). Il faut cependant tenir compte du contexte d'exécution. Cette stratégie se base en effet sur les hypothèses suivantes : chaque noeud est identique, aussi bien sur le plan matériel, logiciel et au niveau des données (utilisation de système de fichiers distribués). Cette hypothèse est nécessaire pour s'assurer qu'une tâche qui migrera sur une autre machine consommera autant de ressources. Ce système est donc adapté aux clusters à haute disponibilité basé sur la réplication de service (comme un cluster de serveur WEB), mais est moins efficace dans les systèmes HPC[5].

3.3.4 Combinaison de stratégies

Il est envisageable d'accumuler certaines approches afin d'économiser plus d'énergie. Il est ainsi possible d'utiliser à la fois l'approche VOVO et IVS ou bien VOVO et CVS. Dans les deux cas, des gains en énergie sont meilleurs, mais ceux-ci sont inférieurs à la somme des gains théoriques. L'accumulation rend les stratégies plus complexe et plus gourmandes en ressources (notamment VOVO-CVS).

[5]High performance Computing

3.3.5 Conclusion

A l'heure actuelle, toute les approches décrites précédemment sont convaincantes et permettent dans certains conditions de réaliser des économies notables. Cependant, les différentes solutions proposées sont soit trop générique et donc moyennement efficaces (IVS par exemple), soit fondées sur des environnement d'exécution très précis et donc difficilement portables sur d'autres architectures. L'approche VOVO notamment nécessite la reconfiguration dynamique de l'architecture du système ainsi qu'un environnement relativement homogène. Ces deux conditions rendent ce type de stratégie inapplicable aux grilles de calculs. Leur architecture est en effet beaucoup trop hétérogène (aussi bien au niveau matériel que système ou même applicatif) pour rendre ce type de stratégie efficace.

Une grille de calcul est pourtant un système consommant beaucoup d'énergie ne serait-ce que par la quantité de machines y participant et l'utilisation de méthodes d'économies d'énergie adaptées aux systèmes massivement distribués est justifiée.

Chapitre 4

La virtualisation

4.1 Présentation

Le concept de machine virtuelle est apparu dans les années 1960-80 avec les mainframes d'IBM. Leur coût élevé rendait inenvisageable l'acquisition de machines personnelles. IBM mit alors en place un système de partage de temps processeur afin de permettre le travail de plusieurs personnes en pseudo-parallèle. Cette solution résolvait en partie le problème, mais une simple erreur de manipulation ou un bug pouvait entraîner l'arrêt de la machine. IBM apporta une solution ultérieurement avec l'architecture VM/370[1] et proposa la virtualisation : une solution permettant le travail à plusieurs sur une même machine en séparant chaque utilisateur sur une instance différente de celle-ci. Une erreur dans une instance n'entrainant pas de problèmes dans les autres.

La baisse des coûts et l'avènement des ordinateurs personnels entraîna un désintérêt pour ce genre de solution, les personnes préférant pour des raisons de performances et de sécurité multiplier les machines afin de séparer les différents utilisateurs et services. Cependant, alors qu'aujourd'hui les machines sont extrêmement puissantes, celle-ci sont de plus en plus sous-exploitées. L'intérêt pour la virtualisation est alors revenu.

[1] Virtual Machine Facility/370

4.2 Architecture

Il est nécessaire de rappeler les bases de l'architecture d'un système classique avant de décrire l'architecture des systèmes virtualisés. L'architecture d'un système peut être vu comme une superposition de trois couches, chacune d'elle utilisant les services de la couche inférieure pour fournir des services plus complexe à la couche supérieure :

- La couche matérielle fournit un ensemble d'instructions simple, appelé ISA[2], lié à l'architecture matérielle (une autre architecture proposera une ISA différente).
- La couche système, utilise ce jeux d'instructions et propose à la couche supérieure un ensemble de fonctionnalitées regroupées dans l'API[3]
- La couche applicative, utilise l'API permettant le développement d'applications pour le système d'exploitation.

Le but d'un système de virtualisation est donc de permettre de simuler au niveau applicatif une nouvelle couche matérielle, identique à la couche inférieure (contrairement à l'émulation qui permet de fournir une ISA différente), afin de pouvoir y greffer un ensemble de machines virtuelles. Ces machines sont complètement isolées du reste du système (espace d'adressage et niveau d'exécution différents).

La virtualisation modifie cette architecture en insérant une nouvelle couche appelée hyperviseur (ou VMM[4]), le plus souvent entre la couche matérielle et la couche système (nous développerons le point du positionnement de l'hyperviseur ultérieurement). Cette hyperviseur fournit à la couche supérieur l'ISA de la couche matériel. Cependant, l'hyperviseur possède un ensemble de mécanismes permettant de faire fonctionner en parallèle plusieurs systèmes, isolés et indépendants les uns les autres. Ces différents systèmes sont appelés machines virtuelles. La figure 4.1 résume cette architecture.

Traditionnellement, un système de virtualisation est principalement caractérisé par son hyperviseur, une machine virtuelle pouvant alors être vue par celui-ci comme un simple flot d'instructions. Les travaux de formalisation sur les machines virtuelles de Popek et Goldberg[20] ont permis de définir le rôle d'un hyperviseur :

- L'hyperviseur doit fournir un environnement d'exécution identique à celui de la machine d'origine. On notera par contre que quelques ex-

[2]Instruction Set Architecture
[3]Application Programming Interface
[4]Virtual Machine Monitor

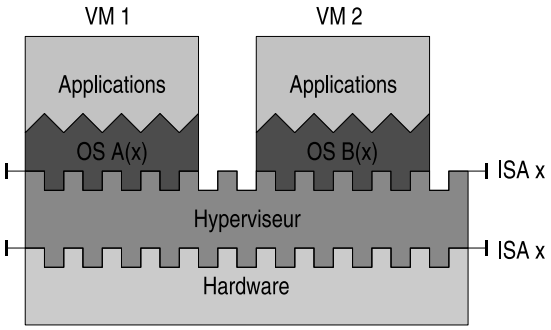

FIG. 4.1 – Architecture pour la virtualisation pure

ceptions apparaissent comme la quantité de ressources disponible (une machine virtuelle disposera toujours de moins de ressources que la machine d'origine), la gestion de l'horloge, ou l'ensemble de périphériques rattachés aux systèmes.
- L'hyperviseur doit pouvoir contrôler la totalité des ressources des machines virtuelles. Celles-ci ne doivent pouvoir accéder à une ressource qui n'aura pas été préalablement allouer par l'hyperviseur.
- Le plus grand pourcentage d'instructions des processeurs virtuels doit pouvoir être traité directement par le processeur, sans intervention de l'hyperviseur.

Il a été dit précédemment que l'hyperviseur se plaçait le plus souvent entre la couche matérielle et les machines virtuelles, substituant la couche système d'origine et si les applications appartiennent toujours au niveau applicatif (pour des raisons de sécurité), le placement des systèmes d'exploitation virtuelles peut varier. La figure 4.2 nous montre les différents placements possibles.

Le schéma 4.2(a) nous montre une implémentation de l'architecture d'un système x86 de type GNU/Linux. Le système d'exploitation fonctionne au niveau 0, niveau où il peut exécuter des instructions privilégiées. Les applications sont quant à elle exécutées au niveau 3, en mode protégé, où l'exécution d'instructions privilégiées entraîne une violation de la protection. Cette architecture permet de protéger le système d'exploitation d'éventuels bugs situés dans la couche applicative.

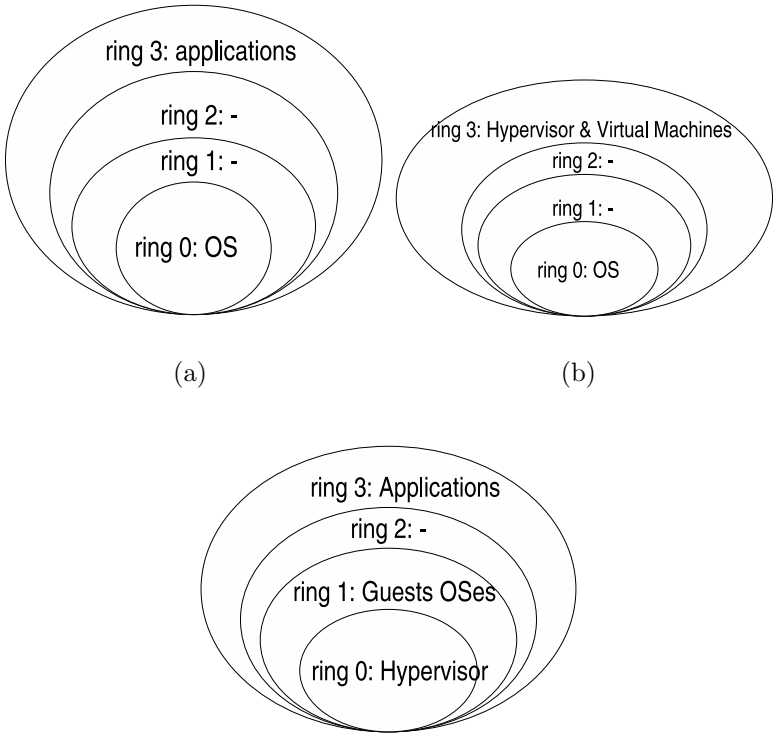

FIG. 4.2 – Différentes séparations des composants d'un système virtualisé

Le schéma 4.2(b) montre l'architecture de solutions comme VMWare ou QEmu. Dans ce cas, tout le système de virtualisation est situé dans le niveau 3. Cette solution permet d'implémenter plus facilement l'hyperviseur puisque celui-ci peut utiliser l'API fournie par le système d'exploitation hôte. L'inconvénient de cette approche est qu'il ne permet pas de séparation simple entre l'hyperviseur, les systèmes d'exploitation des machines virtuelles et leurs applications. L'isolation doit alors être implémentée par les développeurs. Il est également plus difficile d'exécuter des opérations en mode privilégié.

Le schéma 4.2(c) représente l'architecture mise en place pour Xen. Dans ce cas, le système virtuel tournent dans le niveau 1, habituellement non-utilisé, mais pris en compte dans les spécifications de l'architecture x86.

Il reste comme dans le cas classique, dans un niveau d'exécution différent de celui de ses applications. Ce niveau est également en mode protégé et l'exécution d'une instruction privilégiée lève une interruption, récupérée cette fois par l'hyperviseur qui peut alors agir en conséquence. Cette architecture permet de profiter de la sécurité fournie par le matériel pour la protection des différents composants, mais nécessite par contre plus d'expertises pour le développement de l'hyperviseur qui doit fournir des services de plus bas niveau comme les méthodes d'ordonnancement ou l'allocation mémoire.

4.3 Différentes approches

4.3.1 La virtualisation pure

Cette approche fut la première adoptée. Elle consiste à reproduire à l'identique l'ISA fournie par la couche matérielle. Cette approche a l'avantage de ne pas nécessiter d'adaptation au niveau des systèmes d'exploitations des machines virtuelles. Elle a cependant des limites liées entre autres à cette volonté de reproduire exactement l'ISA et à l'existence d'instructions "sensibles" (pouvant mettre à mal la virtualisation ou la protection). On peut en effet, regrouper les instructions d'une ISA (et donc celles reçues par l'hyperviseur) en trois catégories :

- Les instructions non privilégiées et non sensibles. Ces instructions sont considérées comme sûres et ne remettent pas en cause l'isolation ou la protection des machines virtuelles. Elles sont directement exécutées par le processeur dans un niveau protégé (niveau 1 à 3), sans traitement préalable de l'hyperviseur.
- Les instructions privilégiées sensibles. Ces instructions remettent en cause la protection et l'isolation des machines. L'exécution de celles-ci doit se faire en mode privilégié (niveau 0) sinon, une interruption est levée, pouvant être interceptée par l'hyperviseur, qui décidera alors de l'émuler ou non par un autre ensemble d'instructions.
- Les instructions non privilégiées sensibles n'entrainant pas d'interruptions mais remettant en cause la virtualisation. C'est instructions sont les plus délicates à gérer car celles-ci ne génèrent pas d'interruptions et ne sont donc pas directement repérable par l'hyperviseur.

Popek et Goldberg[20] ont défini qu'il n'était possible de virtualiser une architecture que si l'ensemble des instructions sensibles de son ISA appar-

tient à l'ensemble des instructions privilégiées de celle-ci. Les différentes architectures, pour être virtualisables doivent donc dans le meilleur des cas ne pas posséder d'instructions appartenant à la troisième catégorie citée précédemment.

Cependant, la majorité des architectures n'ont pas été spécialement conçu pour la virtualisation. Le processeur x86 (IA-32) par exemple possède dix-sept instructions appartenant à la troisième catégorie[23]. Pour permettre une virtualisation correcte, l'hyperviseur se doit donc de résoudre ces problèmes, en proposant des solutions pouvant impacter sur les performances globales (VMWare[29] modifie les instructions à problèmes reçues par l'hyperviseur en direct puis les stockent dans un cache pour les utilisations futures).

Les changements de contextes permanents, dû à l'exécution en alternance de l'hyperviseur des machines virtuelles a également un impact sur les performances. Il faut en effet à chaque transition, sauvegarder le contexte d'exécution pour le remplacer par le nouveau contexte, précédemment sauvegardé. La TLB[5] pouvant être implémentée de façon relativement basique il peut-être nécessaire de la vider entièrement avant de la re-remplir à nouveau. Ces opérations sont coûteuses et augmentent la charge dû à l'hyperviseur. Une étude d'IBM et de VMWare[3] a pu montrer ainsi un surcoût dû à la virtualisation pure compris entre 12% et 15%.

4.3.2 La para-virtualisation

La para-virtualisation tend à améliorer les performances de l'hyperviseur par rapport à la virtualisation pure. Cette approche est principalement supporté par les équipes travaillant sur Denali[30] et Xen[8].

La virtualisation pure a mis en avant les problèmes de performances qu'entrainait la stricte copie de l'ISA au niveau de l'hyperviseur, notamment l'absence d'optimisations. La para-virtualiasation propose donc de fournir aux machines virtuelles une architecture plus ou moins différentes de l'ISA (cf figure 4.3), dans un but d'optimisation. L'hyperviseur fournira donc une ISA de plus haut niveau, dédiée à la virtualisation, qui lui sera propre.

Le problème du vidage de la TLB à chaque changement de contexte a pu ainsi être résolu. Xen a ainsi choisit de faire fonctionner une partie du code de chaque machine virtuelle en mode privilégié, limitant les change-

[5]Translation Lookaside Buffer : Cache contenant les entrées des tables de pages les plus utilisées

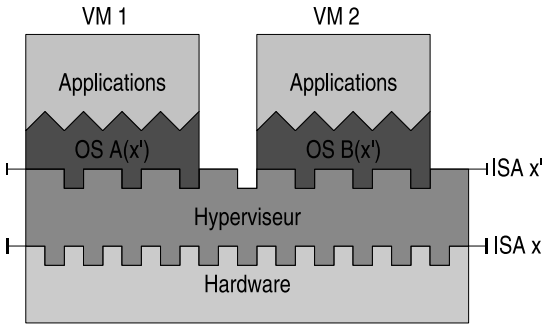

FIG. 4.3 – Architecture pour la para virtualisation

ments de contexte. Le problème des instructions sensibles a été résolu en les remplaçant directement par des appels à l'hyperviseur évitant ainsi la surveillance permanente du flux d'instructions. Denali a également ajouté une nouvelle instruction appelée "*idle-with-timeout*" permettant d'éviter qu'une machine virtuelle ne gaspille trop de cycles CPU en attente active. Xen a décidé de permettre un accès direct en lecture à la mémoire relative à chaque machine virtuelle afin de limiter les accès à l'hyperviseur.

La para-virtualisation a permis un gain de performance non négligeable. Xen obtient de très bonne performances comparable à une exécution native d'un système d'exploitation non modifié puisque différentes évaluations([8, 6]) ont calculé une charge du à la virtualisation d'environ 3%. Ce gain a cependant un coût qui est plus difficilement évaluable : le coût de portage d'un système d'exploitation pour l'hyperviseur.

En effet, la contrainte principale de la para virtualisation vient naturellement de la nécessité d'adapter chaque système d'exploitation pour chaque type d'hyperviseur. Xen a pour objectif de faire tourner un nombre de machines restreint (une centaine) mais pouvant être considérées comme des machines de productions. Afin de faciliter le portage de celles-ci dans les meilleurs conditions, il a été choisi de ne pas trop modifier l'ISA en profondeur. Xen a ainsi pu valider sa solution en modifiant plusieurs systèmes d'exploitation tel que GNU/Linux, FreeBSD ou OpenSolaris. Le portage d'un système GNU/Linux n'ayant nécessité la modification de quatre milles lignes environ. Celui de WindowsXP a du cependant être abandonné, celui-ci ayant

une structure interne trop différente des autres systèmes. Denali a par contre la volonté de faire tourner un grand nombre de machines virtuelles (environ dix mille). Ce choix a nécessité de modifier énormément l'ISA et les systèmes d'exploitations au point où ils ont dû abandonner le portage d'un système existant. La validation de leur contribution se faisant avec un système d'exploitation ultra-léger dédié à leur hyperviseur (Ilwaco).

4.3.3 La pré-virtualisation

La pré-virtualisation[16] tend à réduire, voire à annuler le coût de portage des systèmes d'exploitations pour les hyperviseurs. Pour cela, un greffon est ajouté au système d'exploitation, permettant de réaliser la traduction automatique des instructions de bas niveau du système d'exploitation vers les instructions d'un plus haut niveau de l'hyperviseur. Ce système permet alors de combiner la généricité de la virtualisation pure avec les performances de la para-virtualisation. La figure 4.4 décrit l'architecture de cette approche.

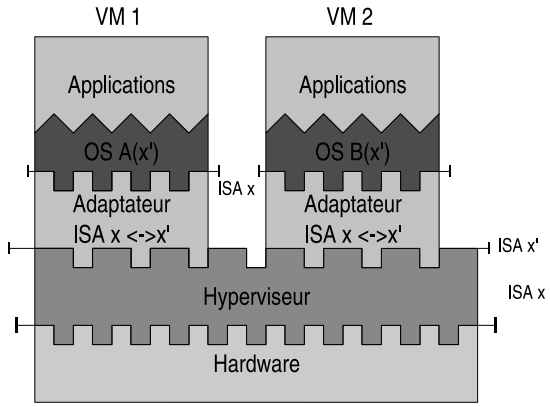

FIG. 4.4 – Architecture pour la pré virtualisation

Le but du greffon est de permettre l'adaptation du système d'exploitation à l'hyperviseur. Cette adaptation nécessite en premier de prendre connaissance des instructions sensibles. Pour ce faire, le système d'exploitation est analysé lorsqu'il est sous la forme de code assembleur. A chaque fois qu'une instruction sensible est détectée, elle est substituée par une forme intermédiaire (qui consiste à insérer un nombre suffisant d'instructions neutres "*nop*". Ces instructions seront écrasées par le véritable code spécifique à

l'hyperviseur ultérieurement). La méthode d'analyse du code assembleur est efficace mais n'est pas parfaite, en effet, certaines instructions ne sont sensibles que dans un certain contexte d'exécution (comme l'instruction *mov* par exemple). Afin de détecter ces instructions, le système est également exécuté dans une machine virtuelle pure qui va le stimuler pour les détecter. Lorsque l'hyperviseur détecte ce type d'instructions, son adresse est sauvegardée.

Le système d'exploitation est ensuite recompilé et les différentes données récoltées lors des phases d'analyse et de compilation sont stockées dans une zone réservée de l'entête ELF. Le choix a été fait de ne pas réaliser la traduction du code au moment de la compilation, afin de ne pas définitivement spécialiser un système d'exploitation pour un type d'hyperviseur. Ne voulant pas non plus pénaliser trop les performances par la traduction, celle-ci se fait lors du chargement du système d'exploitation par l'hyperviseur. Ainsi, si un autre hyperviseur charge le système, une autre traduction sera réalisée.

Si cette approche ne permet pas encore l'adaptation automatique d'un système d'exploitation, elle permet tout de même de réduire la quantité de travail (un noyau linux prévirtualisé pour Xen n'a nécessité que mille lignes à modifier, contre quatre mille avec la para virtualisation). Cette approche ajoute un léger *overhead* par rapport à la para virtualisation mais qui peut être considéré comme acceptable (entre 2 et 3% supplémentaires) puisque les performances restent tout de même meilleur qu'avec une technique de virtualisation pure. L'adaptation au dernier moment des systèmes d'exploitation a permis de valider leur contribution sur quatre types de machines virtuelles différentes (Xen, vNUMA, NOP et L4Ka).

4.3.4 La virtualisation matérielle

La virtualisation matérielle telle que proposée par Intel[22] (avec VT[6] et AMD[1] reconsidèrent les problèmes dû à la virtualisation pure et logiquement, sur les lacunes des processeurs. Les deux entreprises ont donc relancé la création de processeurs dédiés et optimisés pour la virtualisation. A terme, les tâches les plus complexes des hyperviseurs (assurer la protection et l'isolation, le changement de contexte) seront déléguées aux processeurs. L'hyperviseur se concentrera alors sur d'autres domaines comme l'ordonnancement, la gestion des périphériques ou les services supplémentaires (migration par exemple). L'hyperviseur sera optimisé pour les différents processeurs à vir-

[6] *Virtualisation Technology* ou *Vanderpool Technology*

tualisation et collaborera avec eux par le biais de l'ISA.

Les solutions de Intel et AMD sont semblables dans les grandes lignes. Chacunes d'elle propose un nouveau jeu d'instructions dédié à la virtualisation comme des instructions forçant le changement de contexte. Les instructions délicates à virtualiser ont été modifiées. Chaque machine virtuelle possède une structure de données au niveau du processeur (structure VMCS pour Intel et VMCB pour AMD) permettant de désigner son espace d'addressage, de sauvegarder son contexte lors de l'ordonnancement, ou de définir quelles interruptions seront capturées. Un nouveau niveau d'exécution apparaît, le niveau *guest* dédié aux machines virtuelles. Ce niveau assure la protection entre l'hyperviseur qui fonctionne au niveau privilégié 0 (niveau *host*) et les machines virtuelles et permet l'envoi d'interruptions en cas d'exécutions d'instructions privilégiées. Différentes optimisations sont apportées également. AMD par exemple a modifié la structure de la TLB en passant d'une architecture classique à une TLB annotable permettant d'invalider seulement les données de la machine virtuelle en cour d'exécution lors d'un changement de contexte.

Cette technologie permet de revenir à une virtualisation pure. Il n'est plus nécessaire d'adapter chaque système d'exploitation à l'hyperviseur. La virtualisation sera plus sûre (car gérée par la couche matérielle) et plus performante.

4.3.5 L'émulation : Un cas particulier

L'émulation est une approche différente de la virtualisation. Dans les deux cas, on dispose d'une machine virtuelle isolée et protégée ainsi qu'un hyperviseur s'assurant de la gestion des ressources de celle-ci. Cependant, la grande différence entre les deux systèmes vient du fait qu'en émulation, l'ISA proposée par l'hyperviseur correspond à une architecture différente. chaque instruction de la machine virtuelle est interceptée par l'hyperviseur et traduite en une ou plusieurs autres instructions pour l'architecture physique. Ce fonctionnement permet à l'émulation d'avoir des machines virtuelles reposant sur une architecture matérielle différente de celle de l'hyperviseur (Voir figure 4.3.5). Le logiciel Qemu[2] permet par exemple de faire fonctionner sur une architecture x86 ou PowerPC des machines virtuelles utilisant une architecture x86, ARM, Sparc, MIPS ou PowerPC. Ce type de virtualisation permet alors de tester facilement le portage d'applications sur des architectures différentes, sans nécessité d'investissement matériel.

Ce type de virtualisation est par contre peu performant puisque la traduction de chaque instruction est complexe : plusieurs traductions sont possibles, en fonction du contexte et certaines architectures sont tellement différentes que reproduire le fonctionnement exact d'une machine est délicat. Qemu fournit par contre un module accélérateur qui permet lorsque les machines virtuelles et l'architecture matérielle sont de type x86 d'améliorer grandement les performances, il fonctionne dans ce cas plutôt comme une solution de virtualisation pure, en envoyant directement certaines instructions au processeur, sans les modifier.

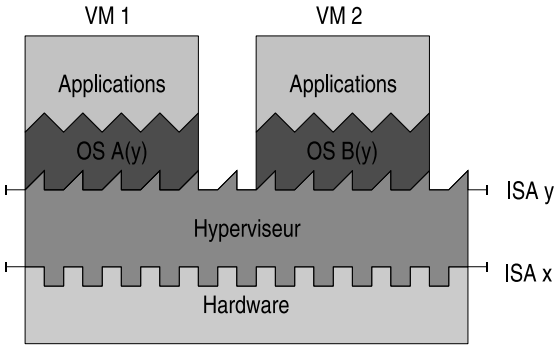

FIG. 4.5 – Architecture pour l'émulation

4.4 Les atouts de la virtualisation

La virtualisation est ainsi aujourd'hui mise en avant dans les solutions de consolidations de serveurs. Les grandes entreprises ont souvent un parc informatique très étendu et des besoins de disponibilités et/ou de sécurités ont fait que chaque service a été déployé sur une machine à part entière. Le principe de consolidation consiste alors à regrouper les services dans des machines virtuelles disposées sur un ensemble plus réduit de machines. Les coûts de maintenance sont alors réduits sans qu'il n'y ait forcément de perte de disponibilité ou de performances.

La virtualisation est également un plus dans le domaine du déploiement. WebSphere de IBM permettait de déployer entre autre des machines vir-

tuelles JAVA à la demande. Ce principe a pu être étendu aux machines virtuelles des hyperviseurs. VMWare et Xen par exemple proposent la migration de machines virtuelles d'un hyperviseur à un autre et ceci, avec où sans arrêt de service. Renato Figuerido et al.[21] mettent ainsi en avant la virtualisation comme une solution permettant de gérer plus facilement les architectures distribuées dynamiques complexes comme les grilles de calculs. Il est ainsi possible de personaliser la grille en fonction d'objectifs précis tel que la diminution du trafic réseau (en regroupant sur un même hyperviseur des machines virtuelles travaillant ensemble) ou l'augmentation des performances (en déplaçant les machines virtuelles nécessitant beaucoup de puissance sur les hyperviseurs possédant le plus de ressources).

Le niveau d'abstraction supplémentaire qu'apporte les machines virtuelles est également un plus pour la supervision ou l'administration d'architectures distribuées. Si un noeud devait être coupé pour une mise à jour quelconque, la migration de ses machines virtuelles sur d'autres noeuds assurerait la disponibilité du service. Les outils de monitoring n'ont plus le besoin d'être porté pour chaque architecture système. Il est juste nécessaire d'adapter l'outil à l'hyperviseur pour que celui-ci surveille toute les machines virtuelles s'exécutant dessus.

Chapitre 5

Conclusion

Les grilles de calculs sont aujourd'hui de plus en plus présente dans le domaine du calcul scientifique. Leur architecture très complexe rends la gestion des évolutions materielles ou leur reconfiguration encore délicate, si l'on souhaite obtenir une bonne qualité de service.

Les progrès réalisés dans le domaine de la virtualisation, aussi bien au niveau des performances (avec la para-virtualisation par exemple) que de services disponibles (avec la migration de machines virtuelles) permettent d'envisager son utilisation dans le domaine des grilles de calculs. Cette approche faciliterait la reconfiguration et augmenterait la souplesse des grilles. Dans ce cas, chaque noeuds disposerait d'un hyperviseur faisant fonctionner les services mis à la disposition de la grille dans une ou plusieurs machines virtuelles. Certains travaux récent comme ceux de K. Keahey et al.[14, 13] montre effectivement l'intêret de l'utilisation de machines virtuelles pour faciliter l'administration et l'extension des grilles.

L'utilisation de méthodes d'économies d'énergies efficace tel que VOVO nécessite la reconfiguration de l'architecture matérielle et explique en partie pourquoi ce type de solution n'existe pas à l'heure actuelle pour les grilles de calculs. Ce type de stratégie repose sur certains pré-requis que les grilles ne peuvent fournir comme la possibilité de répartir ou de concentrer dynamiquement la charge sur les différens noeuds. L'utilisation de machines virtuelles permettrait de développer ce type de service. Dans ce cas, il serait possible de faire migrer dynamiquement des machines virtuelles d'un hôte à un autre, afin de répartir ou de concentrer la charge de travail de la grille. Cela permettrait entre autre d'éteindre les noeuds inactifs et donc d'économiser de l'énergie.

Bibliographie

[1] AMD "Pacifica" Virtualization Technology. `http://enterprise.amd.com/en/Solutions/Consolidation/virtualization.asp%x`, Mars 2005.

[2] Fabrice Bellard. QEMU, A Fast and Portable Dynamic Translator. In *Proceedings of the USENIX 2005 Annual Technical Conference*, pages 41–46, Anaheim, CA, USA, April 2005. USENIX.

[3] Lance Berc, Noshir Wadia, and Stew Edelman. VMWare ESX Server 2 : Performance and Scalability Evaluation. Technical report, IBM, 2004.

[4] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing Energy and Server Resources in Hosting Centres. In *Symposium on Operating Systems Principles*, pages 103–116, 2001.

[5] Guangyu Chen, Konrad Malkowski, Mahmut T. Kandemir, and Padma Raghavan. Reducing Power with Performance Constraints for Parallel Sparse Applications. In *International Parallel & Distributed Processing Symposium*, 2005.

[6] Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews. Xen and the Art of Repeated Research. In *USENIX Annual Technical Conference, FREENIX Track*, pages 135–144, 2004.

[7] Western Digital. Specifications for the 150 GB Serial ATA Raptor X® Clear Cover hard drive. `http://wdc.custhelp.com/cgi-bin/wdc.cfg/php/enduser/std_adp.php?p_faqid%=1406&p_created=1136411689`, March 2005.

[8] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In

Proceedings of the ACM Symposium on Operating Systems Principles, October 2003.

[9] E. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient Server Clusters. In *Proceedings of the Workshop on Power-Aware Computing Systems*, February 2002.

[10] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid : Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150 :1–??, 2001.

[11] Mark Grechanik, Dewayne E. Perry, and Don S. Batory. Using AOP to Monitor and Administer Software for Grid Computing Environments. In *COMPSAC (Computer Software and Application Conference)*, pages 241–248, 2005.

[12] Intel Corporation. Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor White Paper. `ftp://download.intel.com/design/network/papers/30117401.pdf`, march 2004.

[13] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual Workspaces : Achieving Quality of Service and Quality of Life in the Grid. *Scientific Programming Journal 2006*, 2006.

[14] Katarzyna Keahey, Ian T. Foster, Timothy Freeman, Xuehai Zhang, and Daniel Galron. Virtual Workspaces in the Grid. In *Euro-Par*, pages 421–431, 2005.

[15] Eun Jung Kim, Ki Hwan Yum, Greg M. Link, Narayanan Vijaykrishnan, Mahmut T. Kandemir, Mary Jane Irwin, Mazin S. Yousif, and Chita R. Das. Energy Optimization Techniques in Cluster Interconnects. In *International Symposium on Low Power Electronics and Design*, pages 459–464, 2003.

[16] Joshua LeVasseur, Volkmar Uhlig, Matthew Chapman, Peter Chubb, Ben Leslie, and Gernot Heiser. Pre-Virtualization : Slashing the Cost of Virtualization. Technical Report 2005-30, Fakultät für Informatik, Universität Karlsruhe (TH), Novembre 2005.

[17] Maxtor. DiamondMax 11 Datasheet. `http://www.maxtor.com/_files/maxtor/en_us/documentation/data_sheets/dia%mondmax_11_datasheet.pdf`.

[18] Gordon E. Moore. Cramming more Components onto Integrated Circuits. *Electronics*, 38(8), 04 1965.

[19] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Dynamic Cluster Reconfiguration for Power and Performance. In L. Benini, M. Kandemir, and J. Ramanujam, editors, *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, 2002.

[20] Gerald J. Popek and Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Commun. ACM*, 17(7) :412–421, July 1974.

[21] Renato J. Figueiredo and Peter A. Dinda and José A. B. Fortes. A Case for Grid Computing on Virtual Machines. In *ICDCS '03 : Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 550, Washington, DC, USA, 2003. IEEE Computer Society.

[22] Rich Uhlig and Gil Neiger and Dion Rodgers and Amy Santoni and Fernando C.M. Martins and Andrew V. Anderson and Steven M. Bennett and Alain Kägi and Felix H. Leung and Larry Smith. Intel virtualization technology. In *Computer*. IEEE Computer Society, 2005.

[23] J. Robin and C. Irvine. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine monitor. In *Proceedings of the 9th USENIX Security Symposium, Denver*, pages 129–144, August 2000.

[24] Samsung. HD400LJ Specifications. `http://www.samsung.com/Products/HardDiskDrive/SpinPointTSeries/HardDisk%Drive_SpinPointTSeries_HD300LJ.asp?page=Specifications`.

[25] Seagate. NL35 Series Datasheet. `http://seagate.com/docs/pdf/datasheet/disc/ds_nl35series.pdf`, March 2006.

[26] Edmund Smith and Paul Anderson. Dynamic Reconfiguration for Grid Fabrics. In *5th IEEE/ACM International Workshop on Grid Computing*, December, 2004. IEEE/ACM.

[27] Paul Stelling, Ian Foster, Carl Kesselman, Craig Lee, and Gregor von Laszewski. A Fault Detection Service for Wide Area Distributed Computations. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, pages 268–278, Chicago, IL, 28-31 July 1998.

[28] Douglas Thain and Miron Livny. Building Reliable Clients and Servers. In Ian Foster and Carl Kesselman, editors, *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[29] Vmware. `http://www.vmware.com`.

[30] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and Performance in the Denali Isolation Kernel". In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, Dec 2002.