VIRTUALISATION DE SERVEURS LINUX : UNE ÉTUDE COMPARATIVE

par

Fernando Laudares Camargos

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître en sciences (M.Sc.)

FACULTÉ DES SCIENCES


UNIVERSITÉ DE
SHERBROOKE

Sherbrooke, Québec, Canada, 6 décembre 2008

Your file Votre référence
ISBN: 978-0-494-49470-7
Our file Notre référence
ISBN: 978-0-494-49470-7

Le 1er décembre 2008

le jury a accepté le mémoire de M. Fernando Laudares Camargos dans sa version finale.

Membres du jury

M. Gabriel Girard
Directeur
Département d'informatique

M. Richard St-Denis
Membre
Département d'informatique

M. Jean Goulet
Président-rapporteur
Département d'informatique

Sommaire

La virtualisation de serveurs sur l'architecture x86 est, depuis quelques années, un sujet très chaud, qui a mené à des modifications importantes dans la structure de cette architecture ainsi qu'au développement de nouvelles technologies basées sur différentes approches pour implémenter la virtualisation de serveurs.

Nous avons réalisé une étude comparative évaluant l'efficience de six logiciels libres de virtualisation publiés sous la licence GPL : *KQEMU*, *KVM*, *Linux-VServer*, *OpenVZ*, *VirtualBox* et *Xen*. L'efficience des logiciels de virtualisation a été évaluée en fonction de leur performance ainsi que de leur capacité de mise à l'échelle. Notre analyse a été faite en deux étapes. Dans la première étape, nous avons utilisé sept bancs d'essai différents pour évaluer la performance des logiciels de virtualisation et la comparer avec celle de Linux, observant le surcoût imposé par la couche de virtualisation. Ensuite, nous avons évalué la capacité de mise à l'échelle des logiciels de virtualisation avec l'exécution concurrente du même banc d'essai sur 1, 2, 4, 8, 16 et puis 32 machines virtuelles.

Les résultats de la première partie de l'évaluation ont placé Linux-VServer comme le logiciel de virtualisation le plus performant, suivi par Xen. OpenVZ a montré une moins bonne performance que celle attendue d'un logiciel de virtualisation au niveau du système d'exploitation, son point fort étant l'échange de données par réseau. KVM a montré une assez bonne performance, à l'exception de l'utilisation du réseau. VirtualBox a aussi montré une bonne performance pour le transfert de données par réseau, mais une moins bonne performance pour les autres expériences. KQEMU a montré, de manière générale, une piètre performance.

La deuxième partie de l'évaluation a montré que la nature de la charge de travail mise à l'échelle ainsi que le nombre de machines virtuelles exécutant cette charge

affectent la performance générale du logiciel de virtualisation. Ainsi, Linux-VServer obtient une mauvaise performance quand l'expérience utilise Sysbench comme banc d'essai, mais présente une performance proche de celle de Linux pour la mise à l'échelle de la compilation du noyau. Xen présente aussi une bonne performance dans les deux cas tandis que la performance présentée par KQEMU est la pire de toutes, ce qui montre que ce logiciel ne doit pas être utilisé pour la consolidation de serveurs. KVM, OpenVZ et VirtualBox présentent des performances moyennes. Nous n'avons toutefois pas réussi à exécuter l'expérience avec VirtualBox en utilisant 32 machines virtuelles.

Remerciements

Je voudrais remercier mon directeur de recherche Gabriel Girard, qui a accepté le défi de m'avoir comme étudiant, qui a toujours été disponible aux discussions et qui m'a aidé à mener à terme ce travail.

Je remercie Benoît des Ligneris, mon superviseur professionnel, qui m'a ouvert la porte pour ce projet et qui a toujours cru en moi. Je tiens à le remercier de son amitié et de son soutien durant les moments les plus difficiles de ce parcours. Merci d'avoir partagé ta vision d'un monde meilleur ainsi que les dimanches en famille.

Je tiens à remercier toute l'équipe de *Révolution Linux* : cette maîtrise a été réalisé dans le bureau de RL et tous mes collègues y ont participé, directement ou indirectement. Sans leur soutien, leur patience et leur amitié, ce chemin aurait été encore plus difficile. Je remercie particulièrement Guillaume Pratte, Francis Giraldeau et Julien Desfossez, qui m'ont accompagné tout au long de ce processus.

Je remercie l'*Université de Sherbrooke*, à travers le coordonnateur des *Services à la vie étudiante*, Robert Sage, pour m'avoir supporté financièrement à l'aide d'une *Bourse d'exemption des frais supplémentaires pour les étudiantes et étudiants internationaux*, ce qui m'a permis de poursuivre cette maîtrise.

Finalement, je remercie mon épouse, Juliana, pour son amour, son dévouement et son soutien inconditionnels : sans elle à mes côtés, je n'aurais sûrement pas terminé cette maîtrise. Je remercie Laïs, la plus douce fille, qui est née au même moment que ce document. Ses beaux sourires m'ont motivé à finir ce que j'avais commencé. Je remercie aussi mes parents, Laércio et Rose, pour leur amour et pour avoir toujours été là quand j'en avais besoin.

Table des matières

Liste des tableaux

Table des figures

TABLE DES FIGURES

Introduction

L'infrastructure informatique des entreprises est composée de deux classes d'ordinateurs : les ordinateurs de bureau, utilisés par les différents professionnels comme outil de travail, et les serveurs, qui offrent une gamme variée de services informatiques aux utilisateurs, tel que le courriel, l'authentification et l'impression, entre autres.

Au cours des dernières années, la croissance du secteur informatique et l'investissement continu dans la recherche et le développement du matériel ont contribué à rendre les ordinateurs de plus en plus puissants. En revanche, les besoins en ressources de la majorité des logiciels applicatifs qui s'exécutent sur les serveurs n'ont pas connu la même croissance, ce qui fait que la plupart des serveurs qui composent les centres de données des entreprises sont sous-utilisés [22]. Il est difficile de préciser la valeur exacte du taux d'utilisation des serveurs présentement. Les résultats de plusieurs études ([48], [60], [27], [32], [6]) semblent indiquer qu'elle est proche de 15%. Ainsi, toute stratégie ou technologie qui peut améliorer l'utilisation des serveurs sera vue comme un choix intéressant qui devra être considéré par les compagnies [60].

Une solution fréquemment utilisée pour améliorer l'utilisation des serveurs est l'abandon du modèle traditionnel qui associe à un serveur une seule application. Ainsi, plutôt que d'avoir un serveur dédié pour chaque application, on rencontre plusieurs applications qui partagent le même serveur. Présentement, cette approche est encore la plus utilisée car elle est simple à appliquer : quand le besoin d'offrir un nouveau service arrive et que la compagnie n'a pas de nouvelles machines disponibles, l'application est installée « temporairement » sur un serveur déjà en production. À cause de cette croissance dite « organique » un serveur de courriel devient aussi un serveur d'impression ainsi qu'un serveur d'authentification. La principal désavantage de cette approche est qu'un problème ou un mauvais comportement d'une application

ou service peut compromettre le fonctionnement du serveur au complet. De plus, il existe aussi des situations où il est impossible d'installer deux versions différentes d'une même application sur le même système d'exploitation, comme c'est le cas du système de gestion de base de données PostgreSQL [61].

Une autre approche qui vise à améliorer l'utilisation des ressources informatiques est l'emploi d'une technique appelée *virtualisation de serveurs*, ou tout simplement *virtualisation*. Cette technologie permet de faire fonctionner plusieurs systèmes d'exploitation en même temps sur un même serveur, chacun d'eux s'exécutant dans un environnement virtuel distinct. Ceci permet d'installer une seule application par « serveur virtuel », ce qui crée l'illusion qu'il existe un serveur dédié par application. La virtualisation rend possible la consolidation en un seul serveur de plusieurs serveurs sous-utilisés et en même temps assure l'isolation entre les applications. Par conception, un système d'exploitation s'attend à gérer exclusivement toutes les ressources matérielles de l'ordinateur et à être le seul à pouvoir dialoguer directement avec l'UCT [34]. Le rôle des logiciels de virtualisation est de contourner cette problématique : ils doivent faire l'interface entre l'ensemble des systèmes d'exploitation fonctionnant en parallèle et la couche matérielle (composée par des périphériques tels que l'UCT, la mémoire et le disque). Pour ce faire, ils doivent simuler autant de « machines virtuelles »[1] que de systèmes d'exploitation. Il est communément admis que la virtualisation permet l'utilisation des ressources d'un serveur à un taux voisin de 75% [60, 32]. La figure 1 représente un scénario où quatre serveurs sous-utilisés ont été consolidés en un seul à l'aide de la virtualisation ; les barres à côté des icônes représentant les serveurs démontrent l'utilisation des ressources physiques des machines.

[1] On rencontre parfois dans la littérature les termes « machine virtuelle » et « serveur virtuel » utilisés pour faire référence à la même chose, soit un environnement virtuel d'un ordinateur virtualisé. Dans ce travail, nous faisons une distinction entre ces deux termes. Par définition, un serveur est un ordinateur (ou une machine) configuré avec un ensemble de logiciels de façon à offrir un service ou des services à des utilisateurs, qui peuvent être des humains ou encore d'autres ordinateurs, appelés « clients », généralement à travers un réseau. Un serveur virtuel est, alors, une machine virtuelle personnalisée qui offre un service particulier aux clients d'une infrastructure informatique.

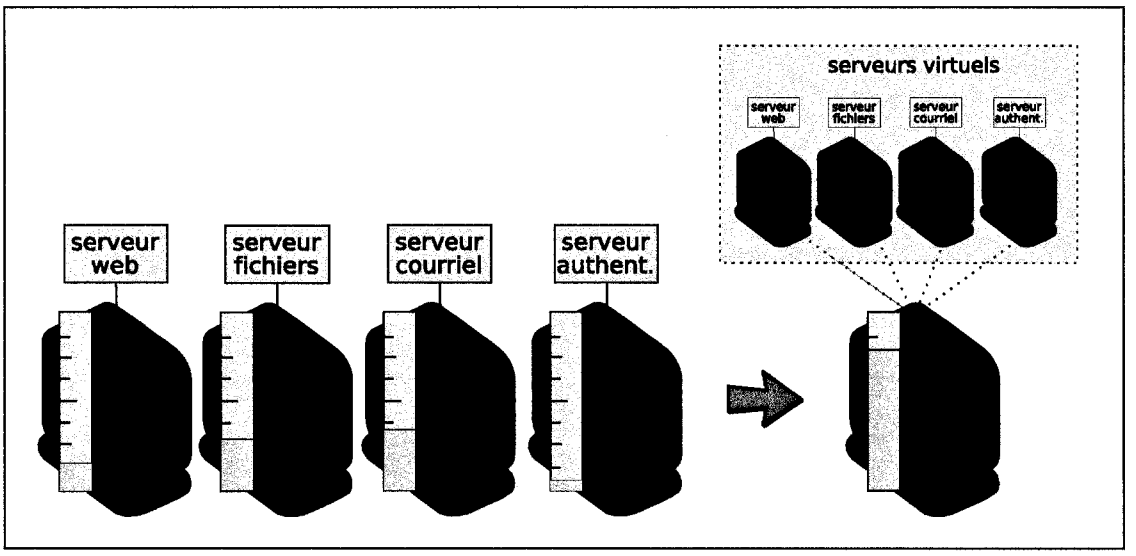

FIG. 1: Consolidation des serveurs à l'aide de la virtualisation

Introduction à la virtualisation

En informatique, le terme *virtualisation* est utilisé dans plusieurs contextes différents, pourtant il fait toujours allusion à une situation d'abstraction des ressources informatiques. Essentiellement, la virtualisation permet au côté logiciel de se détacher du côté matériel [32], éliminant certaines barrières physiques imposées par le matériel. Un exemple illustrant bien cette situation est l'utilisation de la mémoire virtuelle par les systèmes d'exploitation modernes qui permet aux processus d'utiliser une quantité de mémoire plus grande que celle offerte par l'ordinateur [42] en utilisant une partie de l'espace disque pour enregistrer des données.

Le terme virtualisation est aussi employé dans le sens d'agrégation de ressources, par exemple les grappes de calcul, ou plusieurs ordinateurs sont connectés ensemble et gérés par logiciel de façon à créer un seul environnement de travail qui est souvent représenté par une gigantesque et puissante « machine virtuelle ».

Les programmes écrits pour des langages de programmation comme Java, Python et .NET ne sont pas compilés pour une architecture d'ordinateur particulière mais interprétés et exécutés par une *machine virtuelle.* Cette caractéristique rend ces langages compatibles avec toutes les plateformes pour lesquelles les machines virtuelles

respectives existent.

Le contexte abordé par ce travail est celui de la *virtualisation des serveurs*. L'Office québécois de la langue française définit le terme *virtualisation de serveurs* comme *« l'ensemble des techniques qui permettent de diviser les ressources d'un serveur afin de leur faire exécuter séparément et plus efficacement différentes tâches spécifiques, comme s'il agissait de plusieurs serveurs physiques séparés »* [44]. On appelle ces derniers *serveurs virtuels*. Une caractéristique importante des technologies de virtualisation de serveurs est la capacité de faire exécuter sur un serveur virtuel les mêmes applications prévues pour être exécutées sur un serveur physique [25] : un environnement virtuel est vu par une application logicielle comme un environnement réel et ce même si les mécanismes sous-jacents qui le composent sont formellement différents [42].

La virtualisation de serveurs impose un changement dans l'administration de la salle des machines. La perception d'un administrateur de systèmes, lors d'une entrevue publiée dans *Direction Informatique* concernant les effets causés par la virtualisation des serveurs dans la gestion de leur parc informatique, indiquait que cela imposait une autre façon de voir les choses : *« il ne faut plus voir chaque machine en tant que machine, mais en tant que charge de travail, sans quoi on ne peut pas vraiment profiter des avantages qu'offre la virtualisation »* [3].

Tendances

Une enquête réalisée en 2006 par la compagnie spécialisée en études de marché *Forrester Research* auprès de 1221 entreprises situées à différents endroits dans le monde indiquait que, de façon globale, 75% d'entre elles étaient au courant des technologies de virtualisation, 26% avaient déjà déployé des technologies de virtualisation et 8% avaient l'intention de le faire dans les prochains mois [17]. Cette tendance favorable à l'adoption des technologies de virtualisation est confirmée par une autre enquête qui fut réalisée quelques mois plus tard par *Info-Tech Research Group*, une autre compagnie spécialisée en études de marché, auprès de 1500 professionnels de TI, qui rapportait que 80% des compagnies sondées avec plus de 5000 employés avaient adopté ou avaient l'intention d'adopter la virtualisation [37].

L'étude réalisée par *Info-Tech* montrait que, en théorie, il était moins probable que les petites entreprises adoptent la virtualisation : si plus de la moitié (55%) des entreprises sondées qui comptaient de 101 à 500 employées avaient adopté la virtualisation ou avaient l'intention de le faire, seulement un quart (25%) des entreprises avec 100 employés ou moins comptaient le faire [37]. Pourtant, en analysant la composition de l'infrastructure informatique d'un sous-ensemble des compagnies sondées, afin d'évaluer leur degré d'adoption des technologies de virtualisation, il a été découvert que les grandes compagnies, malgré un plus grand taux d'adoption, virtualisent en moyenne seulement 10% de leur infrastructure. La justification donnée pour un si bas taux d'adoption de la virtualisation est attribuée à la complexité politique des grandes compagnies et à leur aversion aux risques. À l'inverse, même si le taux d'adoption de la virtualisation est plus bas en moyenne pour les entreprises de plus petite taille, quand le moment de considérer la virtualisation arrive, elles optent souvent pour une approche du genre *« tout ou rien »* [8] et sont prêtes à confier toute leur infrastructure à la virtualisation. Cela s'explique en partie parce que leur environnement est plus simple et facile à gérer comparé à celui des grandes entreprises [27]. Ceci s'explique également parce qu'elles ont un processus de prise de décision plus simple et plus léger (avec moins de contraintes) et se permettent plus de risques.

Les compagnies qui adoptent la virtualisation à fond voient leur coût d'achat de matériel informatique diminuer de 40 à 75% et épargnent de 25 à 50% par mois en frais de manutention pour ce matériel [8].

La compagnie *Thomas Weisel Partners* a estimé qu'au début de 2006 moins de 4% des serveurs existants basés sur l'architecture x86 avaient été virtualisés [32]. Leur estimation mentionnait qu'approximativement 6% des serveurs vendus en 2005 avaient été virtualisés et leur projection était de 7% pour 2006, confirmant la tendance actuelle pro-virtualisation.

Pourquoi virtualiser l'infrastructure informatique ?

La consolidation des parcs informatiques afin de parvenir à un meilleur taux d'utilisation des serveurs est citée dans la littérature comme étant un des principaux avantages offerts par la virtualisation [60, 32, 22, 42]. Même si la consolidation des serveurs

constitue un avantage intéressant il n'est pas toujours vrai que cela se traduit directement en une économie d'argent. L'utilisation de la virtualisation ajoute une couche de gestion additionnelle à l'administration d'un parc informatique, ce qui demande des administrateurs possédant une formation plus spécialisée. De plus, quand on parle des logiciels de virtualisation propriétaires, il y a des frais d'utilisation (licences) à considérer. Si on ajoute à cette liste le fait que le prix des ordinateurs continue à chuter suivant la « Loi de Moore » [39] et de la forte compétition entre les fabricants, une économie d'argent résultant directement de la consolidation des serveurs à l'aide de la virtualisation, lorsqu'existante, peut se montrer assez limitée.

Pourtant, la virtualisation présente une série d'autres avantages et bénéfices additionnels et peut constituer la solution à divers problèmes, comme :

- un environnement sécuritaire pour le développement, le débogage et l'exécution d'applications ;
- la migration en cours de fonctionnement ;
- la relève rapide de serveurs ;
- la gestion contrôlée des ressources informatiques.

Une des premières sections du chapitre 1 présente d'autres arguments favorables à la virtualisation des infrastructures informatiques et ce dans le cadre de solutions à certains problèmes spécifiques.

Motivation et objectifs de la maîtrise

Il existe plusieurs logiciels conçus pour virtualiser un serveur. Ceux-ci peuvent être classifiés en catégories selon la technologie qu'ils utilisent pour offrir un environnement virtualisé. Les logiciels qui font partie d'une même catégorie partagent plusieurs caractéristiques et, de façon générale, offrent les mêmes possibilités et possèdent les mêmes limitations.

Plusieurs études évaluant ou comparant les différentes technologies de virtualisation ont déjà été publiées. Si celles-ci sont d'une grande valeur pour la compréhension du sujet de cette maîtrise, aucune ne couvre les principaux logiciels libres de virtualisation côte à côte et seulement quelques-unes ont été publiées par des sources indépendantes. La principale raison pour ce manque de couverture est le fait qu'une

telle étude demande beaucoup de temps à être réalisée et cela en fonction du nombre d'expériences qui doivent être exécutées. Pourtant, les résultats d'une étude comparative visant les principaux logiciels libres de virtualisation sont d'un grand intérêt pour toute la communauté informatique et devient le principal aspect motivateur de ce travail.

Le principal objectif de cette maîtrise est d'évaluer l'efficience des principaux logiciels libres de virtualisation pour Linux publiés sous la licence GPL (de l'anglais *General Public License*). Par efficience, on entend la capacité des logiciels de virtualisation à gérer les ressources physiques de la machine et à les partager entre les environnements virtuels en exécution. Cette caractéristique doit amener à un plus grand taux de consolidation des serveurs et, par conséquence, à une meilleure utilisation des ressources informatiques.

Nous avons choisi de limiter l'envergure de notre étude aux logiciels de virtualisation libres pour deux raisons :

1. La nature « ouverte » de cette licence n'empêche pas la publication des résultats d'éventuelles études réalisées avec ces logiciels ou exige qu'une analise minutieuse des résultats soit faite avant qu'une authorisation de publication soit émise.
2. L'accès au code source du logiciel permet de l'adapter si cela s'avère nécessaire de manière à le faire fonctionner dans l'environnement où les expériences sont exécutées, une caractéristique qui s'est déjà montrée utile dans d'autres situations.

Un objectif secondaire de cette étude est d'acquérir une meilleure connaissance des logiciels libres de virtualisation ainsi que de leur mode de fonctionnement. Nous espérons que les résultats de cette étude puissent contribuer à leur développement continu de manière positive.

Nous n'avons pas considéré dans cette étude l'évaluation des particularités ni des forces et des faiblesses des logiciels de virtualisation de manière individuelle. Les principales caractéristiques des logiciels de virtualisation ont déjà été présentées dans la littérature [31, 61, 7].

Organisation du travail

Ce document est organisé comme suit. Le chapitre 1 présente une vision approfondie de la virtualisation des serveurs, des premiers travaux réalisés à la fin des années 1960, en passant par les concepts de base jusqu'aux modifications récentes apportées à l'architecture x86 pour ajouter un support matériel à la virtualisation.

Le chapitre 2 décrit les technologies utilisées par les différents logiciels pour réaliser la virtualisation des serveurs et classifie les logiciels de virtualisation en catégories en fonction de leurs caractéristiques communes. Le chapitre 3 présente d'autres études sur le sujet publiées entre 2003 et 2007, mettant en évidence les différentes charges de travail utilisées dans l'évaluation des logiciels de virtualisation. Il présente aussi un exposé général des bancs d'essai conçus spécifiquement pour l'évaluation des environnements virtualisés.

La méthodologie utilisée dans notre étude comparative est présentée dans le chapitre 4. Le chapitre 5 contient les résultats obtenus. La discussion des résultats est faite au fur et à mesure qu'ils sont présentés.

La conclusion présente un résumé du travail avec une emphase sur ce que nous avons pu conclure de l'efficience des logiciels de virtualisation évalués. Elle contient aussi des suggestions pour des travaux futurs.

Chapitre 1

Vision approfondie de la virtualisation

La puissance des ordinateurs modernes associée aux avantages de la consolidation des serveurs, entre autres, font de la virtualisation une technologie très populaire aujourd'hui. Pourtant, il ne s'agit pas d'une nouvelle idée. Les racines de la virtualisation remontent à l'époque des ordinateurs « mainframe » IBM des années 1960 et des techniques de partage du temps de traitement (ou « temps partagé », de l'anglais *time-sharing*) [27].

Malgré son succès dans le passé, la virtualisation ne fait pas partie du projet original de conception de l'architecture x86, débuté à la fin des années 1970 et qui est devenue l'architecture la plus populaire du marché. Les architectes en charge du projet ne s'attendaient pas à ce que cette architecture, conçue pour les premiers micro-ordinateurs personnels, puisse un jour être utilisée par des serveurs. Aujourd'hui, les deux principaux fabricants de processeurs x86 font des efforts pour contourner les faiblesses de cette architecture de manière à mieux profiter des avantages de la virtualisation, une technologie qui existe depuis plus de quatre décennies.

Ce chapitre rapelle l'origine de la virtualisation des serveurs et présente en détail le travail de Popek & Goldberg [50], considéré encore aujourd'hui comme une des principales références sur le sujet. Les caractéristiques de l'architecture x86 sont aussi présentées, avec un accent sur les nouvelles extensions incorporées récemment par les fabricants de processeurs de manière à rendre cette architecture « virtualisable ».

Finalement, des arguments favorables à la virtualisation des serveurs sont présentés à la fin du chapitre.

1.1 L'origine des machines virtuelles

Dans la première moitié des années 1960, le centre de recherche d'IBM Thomas J. Watson à Yorktown Heights, New York, a accueilli le projet M44/44X, lequel avait comme objectif d'évaluer les concepts utilisés dans les systèmes à temps partagé en les mettant en pratique et en les mesurant [13].

Le concept de partage du temps de traitement des ordinateurs (*time-sharing*) a commencé à apparaître à la fin des années 1950. Son but était d'améliorer la performance des systèmes informatiques de l'époque en proposant un accès multiple et concurrent aux ordinateurs « mainframe » et de mieux utiliser les ressources de ce matériel qui était assez dispendieux [42]. À cette époque, l'utilisation des ordinateurs se faisait selon le concept de traitement par lots (*batch processing*) : les utilisateurs soumettaient leurs travaux aux ordinateurs à travers des cartes perforées, lesquelles étaient exécutées en séquence. Chaque utilisateur devait donc attendre l'arrivée de son tour pour que son travail soit traité par l'ordinateur. Les systèmes à temps partagé offrent la possibilité aux utilisateurs d'interagir directement avec l'ordinateur (et de manière concurrente entre eux) au travers d'un terminal et de recevoir immédiatement les résultats des calculs [70].

L'ordinateur expérimental utilisé dans le projet M44/44X était une version modifiée du IBM 7044 (le « M44 »). Cet ordinateur était capable de simuler plusieurs instances (ou images) de lui-même, appelées « 44X ». Chacune de ces instances offrait une émulation partielle de l'ordinateur M44. Chaque utilisateur n'accédait pas la machine réelle directement mais une de ces instances et cela en exclusivité. Dave Sayre, le chef du projet, a baptisé ces instances *« virtual machines »* (machines virtuelles) [65].

À cette même époque, le premier système de temps partagé utilisé en production, appelé *Compatible Time-Sharing System* (CTSS), est né au *Massachusetts Institute of Technology* (MIT) [11]. Le CTSS a été développé sur une plateforme similaire à celle du projet M44/44X, l'IBM 704 et ses successeurs (709, 7090 et 7094). Il fonctionnait

avec un programme superviseur[1] qui possédait un contrôle direct et total sur l'accès à toutes les ressources physiques de l'ordinateur [58], une caractéristique essentielle pour la construction de machines virtuelles (presentée en détails dans la Section 1.2.3).

En 1964, IBM introduit sur le marché le fameux System/360, aussi connu comme S/360, une nouvelle catégorie d'ordinateurs qui a remplacé les anciennes plateformes IBM, comme la série IBM 7000 [70]. Le S/360 promettait être rétro-compatible avec les plateformes périmées : plusieurs modèles de la série S/360 offraient la possibilité d'exécuter des logiciels écrits pour les anciennes plateformes par l'émulation de micro-code. Pourtant, la principale innovation technologique qui a accompagné le S/360 était le système CP/CMS, la deuxième génération des systèmes à temps partagé [11]. Le système CP/CMS a été fortement inspiré par le système CTSS, mais sa conception contenait une différence importante : la gestion des ressources physiques de l'ordinateur était séparée de la partie qui offrait le support aux utilisateurs. Elles étaient conçues par des composants différents : le logiciel de contrôle (de l'anglais *control program* - CP), qui jouait le rôle de superviseur et gérait toutes les ressources du système, et le système de moniteur conversationnel (traduction libre de *conversational monitor system* - CMS), responsable d'offrir les environnements interactifs avec lesquels les utilisateurs partageaient l'accès à l'ordinateur [56] (similaire au 44X, mentionné précédemment).

Le logiciel de contrôle, CP, était dérivé du système CP-40 développé à l'université de Cambridge. L'équipe qui a développé le système CP-40, un logiciel de contrôle conçu pour une version légèrement modifiée [2] du modèle 40 du S/360, le S/360-40, a décidé que la façon la plus sécuritaire de protéger les utilisateurs les uns des autres n'était pas de leur offrir à chacun une image de la machine réelle, comme faisait le M44/44X, mais plutôt une version complète de la machine (dans ce cas, le S/360). Cet environnement particulier était appelé une « pseudo-machine » et il se différenciait de la « machine virtuelle » 44X par sa capacité à simuler (ou « virtualiser ») l'entièreté

[1] À ce moment historique de l'informatique, les systèmes d'exploitation étaient connus comme *des superviseurs* [29].

[2] L'utilisation de la mémoire associative a été ajoutée au S/360 pour aider le processus de traduction dynamique des adresses mémoire [11] et permettre l'utilisation du concept de *mémoire virtuelle*, développé à l'Université de Manchester à la fin des années 1950 pour permettre l'exécution de logiciels dont la taille était plus grande que la mémoire physique disponible dans la machine [57].

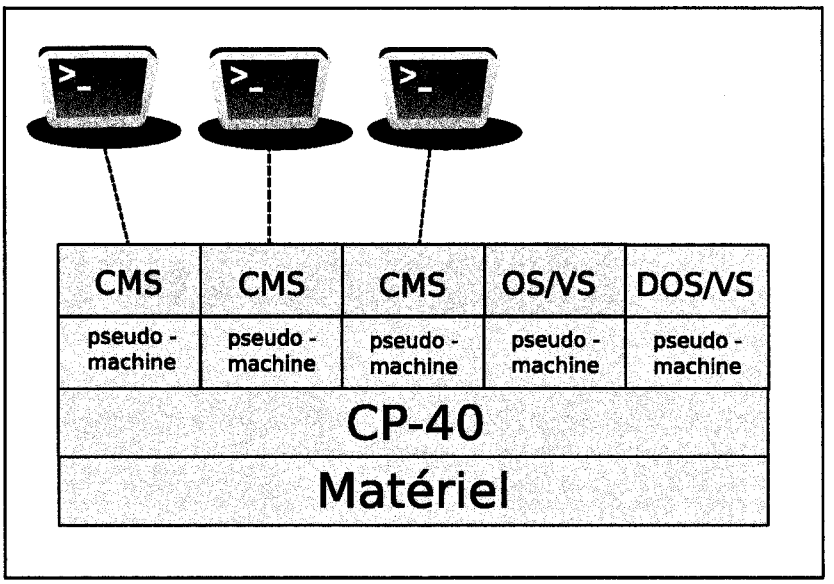

FIG. 1.1: Le logiciel de contrôle CP-40 installé sur un mainframe et configuré avec cinq pseudo-machines

des couches matérielles de la machine physique, tandis que le M44/44X ne simulait qu'une partie du matériel [65]. Pour cette raison, le CP-40 est considéré comme le premier système d'exploitation à supporter le concept de machines virtuelles par la virtualisation complète du matériel sous-jacent [70].

Le système conversationnel CMS, de son côté, était mono-usager en ce sens qu'il ne supportait que l'accès à un seul terminal et, par conséquence, un seul utilisateur pouvait l'accéder à la fois. Comme le système CP était capable de supporter plusieurs systèmes CMS, un par pseudo-machine, la solution utilisée pour supporter plusieurs utilisateurs consistait à fournir une pseudo-machine par utilisateur[3] [56]. La figure 1.1, adaptée du travail de Seawright [56], présente le schéma d'un système IBM System/360 avec le logiciel de contrôle CP-40 et 5 pseudo-machines : un environnement CMS a été installé sur trois d'entre elles, auxquelles se connectent trois terminaux, et les deux autres accueillent, au lieu de CMS, deux autres systèmes d'exploitation compatibles avec l'architecture de la machine.

Même si le M44/44X et le CP-40 étaient deux projets très similaires, ils ont été

[3]Le CP-40 supportait jusqu'à 14 pseudo-machines [65] et conséquemment le même nombre d'utilisateurs simultanément.

développés de manière complètement indépendante. Le terme pseudo-machine, utilisé par l'équipe du CP-40, a été remplacé par machine virtuelle quand R. J. Creasy, développeur-chef du CP-40, a pris connaissance du projet M44/44X lors d'une visite au centre de recherche d'IBM [65].

Le CP-40 a été utilisé uniquement dans la machine expérimentale S/360-40 à l'Université de Cambridge et n'a jamais été mis en production. Le modèle 67 du S/360 a incorporé les modifications matérielles proposées dans le S/360-40 afin de permettre l'utilisation du concept de mémoire virtuelle [57], nécessaire pour l'exécution des machines virtuelles, et a été mis en production accompagné d'une nouvelle version du logiciel de contrôle, le CP-67. Le S/360-67 est considéré comme le premier ordinateur commercial à implanter la virtualisation [71].

1.2 Moniteur de machines virtuelles

Le logiciel, ou la couche logicielle, responsable de l'exécution des machines virtuelles ainsi que de la gestion des ressources physiques de la machine est appelé moniteur de machines virtuelles, ou VMM (de l'anglais *Virtual Machine Monitor*). Son rôle est similaire à celui d'un système d'exploitation ordinaire, mais plutôt que de répondre aux demandes des processus, il doit gérer les demandes des machines virtuelles. Cette section présente le travail de Popek & Goldberg, une référence importante dans le domaine de la virtualisation de serveurs, ainsi que le mode de fonctionnement et les différents types de VMMs.

1.2.1 Le travail de Popek & Goldberg

En 1974, Gerald J. Popek et Robert P. Goldberg ont publié un article sur la virtualisation [50] qui est considéré encore aujourd'hui comme une des principales références sur le sujet et qui a été utilisé comme base pour le développement des technologies de virtualisation, présentées dans le chapitre 2.

Dans cet article, ils définissent formellement les trois propriétés essentielles afin qu'un logiciel de contrôle puisse être considéré un VMM :

1. *il (le VMM) doit offrir un environnement d'exécution « essentiellement identique » à celui de la machine physique ;*
2. *les logiciels qu'il exécute doivent exhiber, au pire, une légère perte d'efficience ;*
3. *il (le VMM) doit avoir le contrôle complet des ressources du système.*

La première propriété fait référence à l'*équivalence* (ou la transparence) d'exécution : tout logiciel exécuté dans une machine virtuelle doit présenter les mêmes effets que s'il était exécuté directement sur la machine réelle. Il existe, cependant, deux exceptions à cette règle :

- La différence des ressources disponibles : si la machine virtuelle possède moins de ressources (par exemple, mémoire) que la machine physique, le logiciel peut s'exécuter différemment[4].
- Les dépendances de synchronisme : certaines instructions peuvent prendre plus de temps à s'exécuter dans un environnement virtuel à cause de l'intervention du VMM (cette situation est présentée en détails dans la Section 1.2.3).

La deuxième propriété fait référence à l'*efficience* : statistiquement, la majorité des instructions envoyées aux processeurs virtuels doivent être traitées directement par le processeur physique, sans que le VMM n'intervienne. Seulement les instructions sensibles (voir les explications à la Section 1.2.2) doivent être interprétées par le VMM. Cette propriété vise à différencier les VMM des émulateurs qui eux interprètent la totalité des instructions et qui, pour cette raison, sont moins efficients (exhibant une sous-performance considérable).

Finalement, la troisième propriété fait référence à la *sécurité* [1] en attestant que le VMM doit garder le contrôle sur toutes les ressources physiques de la machine, comme la mémoire. Cette propriété est renforcée par deux contraintes :

1. un logiciel qui s'exécute dans une machine virtuelle ne doit pas pouvoir accéder à une ressource qui n'a pas été allouée à cette machine virtuelle par le VMM ;
2. dans certaines conditions, le VMM doit pouvoir récupérer le contrôle des ressources déjà allouées à une machine virtuelle.

[4]Une façon d'illustrer cette situation est de voir l'environnement d'une machine virtuelle comme une version réduite de celle de la machine physique : ils sont identiques sous le point de vue logique mais l'environnement virtuel a à sa disposition une fraction des ressources physiques de la machine.

Ce célèbre article de Popek & Goldberg, malgré sa forme généraliste, ciblait la virtualisation d'une architecture particulière, l'architecture dite de troisième génération[5], et proposait trois théorèmes, le principal d'entre eux étant le suivant :

> "*Il est possible de construire un moniteur de machines virtuelles pour tout ordinateur conventionnel de troisième génération si l'ensemble des instructions sensibles de cet ordinateur est un sous-ensemble de l'ensemble des instructions privilégiées.*"

Pour interpréter ce théorème, il faut d'abord connaître les différents types d'instructions qui composent le jeu d'instructions élémentaires et les modes d'opération d'un processeur.

1.2.2 Le jeu d'instructions élémentaires et les modes d'opération d'un processeur

Le jeu d'instructions élémentaires qui compose l'architecture d'un processeur est identifié par l'acronyme anglais *ISA* pour « *Instruction Set Architecture* ». Ce jeu d'instructions est stocké en mémoire non volatile et définit les capacités du processeur, dont l'architecture matérielle est optimisée de manière à exécuter les instructions de l'ISA le plus efficacement possible [34].

Pour les besoins de la virtualisation, les instructions peuvent entre classées en deux catégories en fonction de leur comportement : les instructions *sensibles* et les instructions *inoffensives (innocuous)*. Les instructions sensibles sont celles qui peuvent changer l'état du système (comme les instructions d'entrée-sortie) ou modifier l'allocation des ressources (manipulant, par exemple, les registres de contrôle du processeur) [42]. Les instructions qui ne sont pas sensibles sont dites inoffensives.

L'architecture des ordinateurs modernes (de troisième génération ou plus récents) définit au moins deux modes d'opération du processeur, chacun avec différents niveaux de privilèges. On comprend par privilège la capacité d'exécuter une instruction. Il n'y a pas de restrictions pour le mode d'opération le plus privilégié, qui peut

[5]La principale caractéristique d'une architecture de troisième génération est l'implantation de deux modes d'opération du processeur, un mode privilégié et l'autre non privilégié, comme présentée dans la section suivante.

exécuter toutes les instructions qui composent l'ISA. Le mode ou les modes d'opération moins privilégiés ne peuvent exécuter qu'une partie des instructions du ISA, soit les instructions considérées par l'architecture comme non privilégiées. Une tentative d'exécuter une instruction privilégiée lorsque le processeur opère dans un des modes moins privilégiés cause une exception.

Le système d'exploitation utilise les modes d'opération du processeur pour implanter un mécanisme de protection pour l'exécution des processus. Généralement, le seul processus exécuté en mode d'opération privilégié est le système d'exploitation lui-même. Quand un autre processus essaye d'exécuter une instruction privilégiée, comme une opération d'entrée-sortie, une exception est générée, donnant au système d'exploitation la possibilité de reprendre le contrôle. Le système d'exploitation peut, alors, exécuter l'instruction demandée par le processus en toute sécurité et lui retourner le résultat de l'exécution de l'instruction. Ce mécanisme de protection assure au système d'exploitation le contrôle total de la machine.

Si on revient au théorème proposé par Popek & Goldberg [50], la condition pour qu'une machine soit virtualisable est que l'ensemble des instructions sensibles disponibles sur le processeur soit un sous-ensemble des instructions privilégiées. Comme nous le verrons plus tard, cette condition est la clé du développement d'un VMM efficient et de fonctionnement simple.

Un exemple d'architecture qui respecte cette condition est celle du IBM System/370. Un exemple d'architecture qui ne respecte pas cette condition est l'architecture x86 utilisée, entre autres, par Intel pour son processeur Pentium 4 et présentée à la Section 1.3.

1.2.3 Mode de fonctionnement du VMM

Le rôle de gestionnaire du VMM est similaire à celui du système d'exploitation, mais plutôt que de gérer l'exécution de processus, il doit gérer l'exécution de machines virtuelles. Plus précisément, il doit gérer les systèmes d'exploitation qui sont exécutés à l'intérieur des machines virtuelles. On appelle ceux-ci des systèmes d'exploitation invités par rapport au système d'exploitation hôte, qui est celui exécuté sur la machine réelle. En effet, le VMM peut être intégré au système d'exploitation hôte ou il peut

être une application exécutée sur ce dernier. Avant de présenter les différents types de VMM nous introduisons deux termes fréquemment utilisés dans la définition des VMM : le superviseur et l'hyperviseur.

Comme le système d'exploitation est responsable de l'exécution des autres applications et qu'il est généralement la seule application à être exécutée en mode privilégié, il est aussi connu sous le nom de *superviseur*. Pour cette même raison, le mode d'opération privilégié du processeur est aussi appelé mode superviseur tandis que le mode d'opération non privilégié (quand il y en a seulement un) est appelé mode utilisateur. Comme le VMM gère l'exécution des superviseurs (les systèmes d'exploitation invités), il est appelé *hyperviseur*.

On rencontre dans la littérature l'emploi des termes VMM et hyperviseur pour désigner le logiciel (ou la couche) de virtualisation. Il existe deux types, ou catégories, d'hyperviseurs. Les hyperviseurs de type I s'exécutent directement sur la couche matérielle [26]. Il s'agit d'un système d'exploitation ou noyau avec des mécanismes pour supporter la virtualisation. Il est responsable de l'allocation et de l'ordonnancement des ressources aux machines virtuelles du système et contient les pilotes nécessaires pour accéder aux périphériques [55]. Le VMM défini par Popek & Goldberg [50] est un hyperviseur de type I.

Les hyperviseurs de type II s'exécutent en tant qu'application sur le système d'exploitation hôte. Ils offrent les services de support à la virtualisation mais sont exécutés en mode non privilégié et reposent, par conséquent, sur le système d'exploitation hôte pour l'allocation et la gestion des ressources matérielles. La figure 1.2 montre un schéma représentant les deux types d'hyperviseurs.

Le VMM défini par Popek & Goldberg [50] est composé de trois modules : le module expéditeur, le module gestionnaire de ressources (traduction libre de l'anglais *allocator*) et un troisième module composé d'un ensemble d'interpréteurs. L'expéditeur est le module de contrôle principal. Il est mis en service chaque fois qu'une machine virtuelle exécute une instruction sensible, laquelle doit être interceptée par le VMM. Selon la nature de l'instruction sensible, le VMM la rédirige vers un des deux autres modules. S'il s'agit d'une instruction qui change potentiellement la configuration des ressources du système, elle est rédirigée vers le module gestionnaire de ressources. C'est lui qui gère les ressources du système et qui assure qu'une même

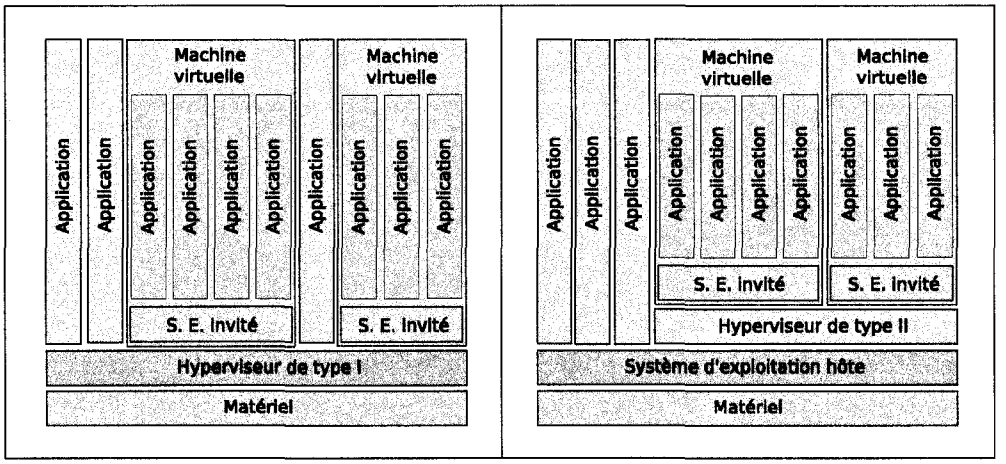

FIG. 1.2: Catégories d'hyperviseurs : hyperviseur de type I, à gauche, et hyperviseur de type II

ressource ne soit pas utilisée simultanément par deux machines virtuelles. Les autres instructions sont redirigées vers le module contenant les interpréteurs. Il existe un interpréteur pour chacune des instructions sensibles, son rôle étant de simuler les effets causés par l'exécution de ces instructions, comme le changement des registres de contrôle du processeur. Cette approche évite que les machines virtuelles aient connaissance de l'état réel des ressources matérielles [55]. En effet, les ressources matérielles sont partagées par les machines virtuelles en exécution et pourtant chaque machine virtuelle croit avoir un accès exclusif à ces ressources et le VMM doit nourrir cette illusion. La figure1.3 illustre le fonctionnement du VMM.

1.3 La virtualisation de l'architecture x86

L'architecture x86 est une des architectures les plus populaires dans l'univers des serveurs, sinon la plus populaire, et pour cette raison est l'objet de notre étude. Pourtant, elle n'a pas été conçue dans le but de supporter la virtualisation [42] et, en conséquence, ne satisfait pas les conditions définies par Popek & Goldberg [50] qui identifient une architecture virtualisable. Plus précisément, 17 des instructions qui composent l'ISA de l'architecture x86 sont des instructions sensibles mais non privilégiées [55]. En conséquence, elles échouent silencieusement au lieu de générer une

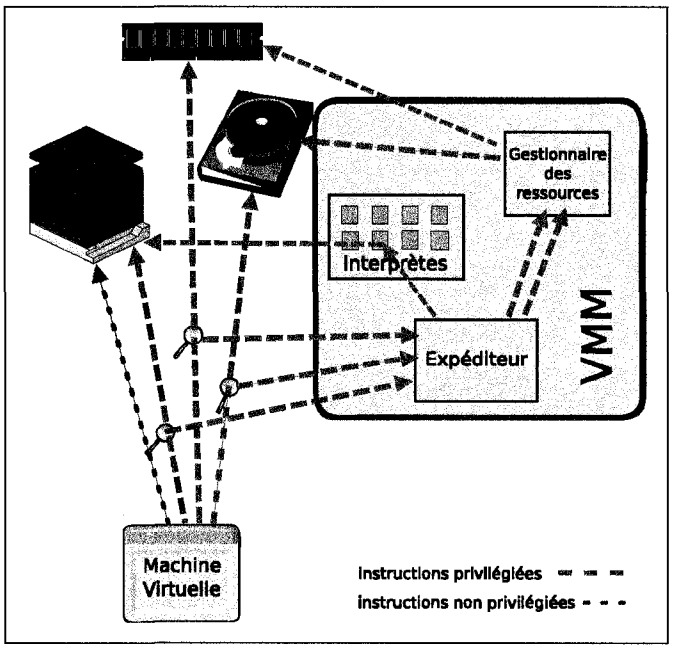

FIG. 1.3: Fonctionnement du VMM, destacant l'interception des instructions privilégiées par le VMM pour qu'elles soient interprétées

exception quand elles sont déclenchées par un processus s'exécutant dans le mode d'opération non privilégié du processeur et cela sans offrir l'opportunité d'être interceptées et interprétées par un VMM.

Pour contourner cette problématique et permettre la virtualisation de l'architecture x86 originale en utilisant des techniques logicielles alternatives, l'hyperviseur doit trouver un moyen de détecter l'exécution de ces instructions sensibles, puis de les interpréter [34]. La seule technique permettant cette détection consiste à intercepter l'exécution de toutes les instructions, privilégiées ou non, pour déterminer si elles sont sensibles, puis de les interpréter si cela est nécessaire. Cette approche entraîne une surcharge de calcul importante qui affecte la performance du système et sa capacité de mise à l'échelle. C'est pourtant une approche utilisée par les VMMs qui emploient la technologie de virtualisation complète sous l'architecture x86 standard, présentée à la section 2.1.

1.3.1 Support matériel à la virtualisation

Récemment deux des principaux fabricants de circuits intégrés au monde, Intel et AMD, ont reconnu l'importance de la virtualisation et, depuis 2005, ont commencé à ajouter du support matériel à la virtualisation dans une partie de leur nouvelle génération de processeurs [23].

Le support matériel est constitué d'une série d'extensions incorporées à l'ISA du processeur, dont l'objectif est, dans un premier temps, de permettre à l'hyperviseur d'intercepter toutes les instructions sensibles exécutées par les applications sans faire appel à des techniques logicielles complexes, et, dans un second temps, d'améliorer encore plus la performance du système en permettant une virtualisation efficiente de la mémoire et des périphériques.

Contrairement à l'ensemble des instructions qui forment la base de l'ISA x86, les extensions utilisées dans le support matériel à la virtualisation ne sont pas standardisées. Celles incorporées aux processeurs AMD sont différentes de celles utilisées par Intel. Pourtant, malgré leur incompatibilité, elles sont essentiellement très similaires et développées autour d'un nouveau mécanisme qui différencie l'environnement d'exécution d'une machine virtuelle de l'environnement d'exécution de la machine hôte. Ce nouveau mécanisme est composé [31] :

- d'un nouveau mode d'opération du processeur utilisé par les instructions exécutées dans une machine virtuelle ;
- d'un système qui utilise deux tableaux pour enregistrer l'état du système, un pour chaque mode d'opération du processeur, de façon à ce que le processeur revient au mode d'opération qu'il avait quitté, le contenu des registres sera le même qu'avant ;
- d'un système qui enregistre la raison du changement du mode d'opération du processeur pour que l'implémentation de l'hyperviseur puisse être plus flexible par rapport à l'action à entreprendre.

AMD a baptisé la technologie utilisée dans ses processeurs pour le support matériel à la virtualisation « AMD-V » (V pour virtualisation). Intel, de son côté, appelle sa technologie *« VT »* (de l'anglais *Virtualization Technology*)[6]. Celle qui est la mieux

[6]La technologie de virtualisation de Intel (VT) est utilisée tant sur les processeurs IA-32 (x86) comme sur les processeurs IA-64 (Itanium). Pour faire la distinction entre les deux, les préfixes x(86)

décrite dans la littérature est celle d'Intel, que nous abordons brièvement avec pour objectif de mieux comprendre les modifications apportées à l'architecture x86 pour permettre la virtualisation matérielle. Nous faisons aussi référence autant que possible à la technologie d'AMD.

Le chapitre 2 présente en détails les différentes techniques utilisées pour virtualiser l'architecture x86, avec ou sans le support matériel à la virtualisation.

1.3.1.1 Virtualisation du processeur

Les processeurs qui utilisent l'architecture x86 standard possèdent quatre modes d'opération représentés en forme d'anneaux sur la figure 1.4. Chaque anneau définit un niveau de privilège décroissant. Le niveau 0 est dédié aux applications les plus critiques, comme le noyau, réclamant les privilèges les plus élevés [34]. Les deux niveaux suivants, 1 et 2, sont réservés aux pilotes des périphériques [42] tandis que les autres applications sont exécutées dans le niveau 3, le moins privilégié des quatre. Pourtant, le seul système d'exploitation commercial moderne à utiliser les niveaux 1 et 2 est OS/2 d'IBM [2]. En général, le système d'exploitation (noyau) et les pilotes des périphériques sont exécutés en niveau 0 et les applications sont exécutées en niveau 3. En pratique, cela fait du x86 une architecture avec deux modes d'opération, un privilégié et un non privilégié.

Le coeur des technologies de virtualisation de AMD et Intel est l'ajout d'un nouveau « niveau » d'opération du processeur composé, lui aussi, de quatre modes d'opération, aussi appelés « *rings* »(anneaux). Dans ce nouveau modèle, le VMM (de type I) est exécuté dans le mode d'opération le plus privilégié (*ring 0*) du niveau d'opération standard, baptisé *« VMX root »* chez Intel et « *host mode* » chez AMD. Pour une question d'uniformité, nous l'appelons « niveau hôte ». Les systèmes d'exploitation invités sont exécutés dans le mode d'opération le plus privilégié (*ring 0*) du nouveau niveau d'opération du processeur, appelé *« VMX non-root »* chez Intel et « *guest mode* » chez AMD [43]. Nous appelons ce niveau « niveau invité ». Les applications exécutées dans le niveau invité sont moins privilégiées que celles exécutées dans le

et i(tanium) sont utilisés, formant ainsi deux branches de la technologie VT, VT-x et VT-i. Comme ce travail se concentre sur la virtualisation des processeurs x86, le terme *VT* fera toujours référence à la technologie *VT-x*.

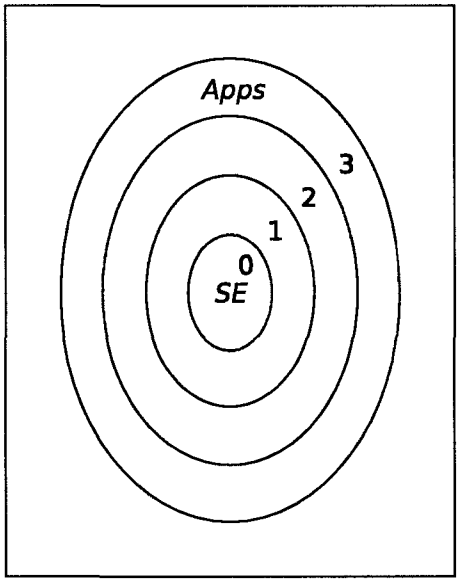

FIG. 1.4: Structure en forme d'anneaux représentant les niveaux de privilèges supportés par l'architecture x86.

niveau hôte et ce indépendamment du mode d'opération dans lequel elles sont exécutées [64]. Cette architecture permet la construction d'un mécanisme de contrôle qui force le passage du processeur au niveau d'opération hôte (et au contrôle du VMM) chaque fois qu'une instruction sensible est exécutée dans le niveau invité. La figure 1.5 présente un schéma de la technologie de virtualisation de Intel.

Quand le contexte l'exige, le processeur bascule d'un niveau d'opération à l'autre [34]. La transition du niveau hôte au niveau invité se fait à l'aide de l'instruction *VM entry* chez Intel et de l'instruction *VMRUN* chez AMD [73]. La transition du mode invité au mode hôte est faite à l'aide de l'instruction *VM exit* chez Intel ; AMD n'utilise pas une instruction spécifique pour ce cas et la transition se fait en fonction de certaines circonstances spéciales[7].

Pour mieux illustrer le mode de fonctionnement de cette technologie, supposons une situation où une application d'une machine virtuelle, s'exécutant dans le mode d'opération le moins privilégié (*ring 3*) du niveau invité, a besoin d'ajouter des don-

[7]Le comportement du processeur est différent quand il fonctionne en niveau invité. Plusieurs instructions et événements forcent le passage inconditionnel au niveau hôte, le contrôle retournant au VMM.

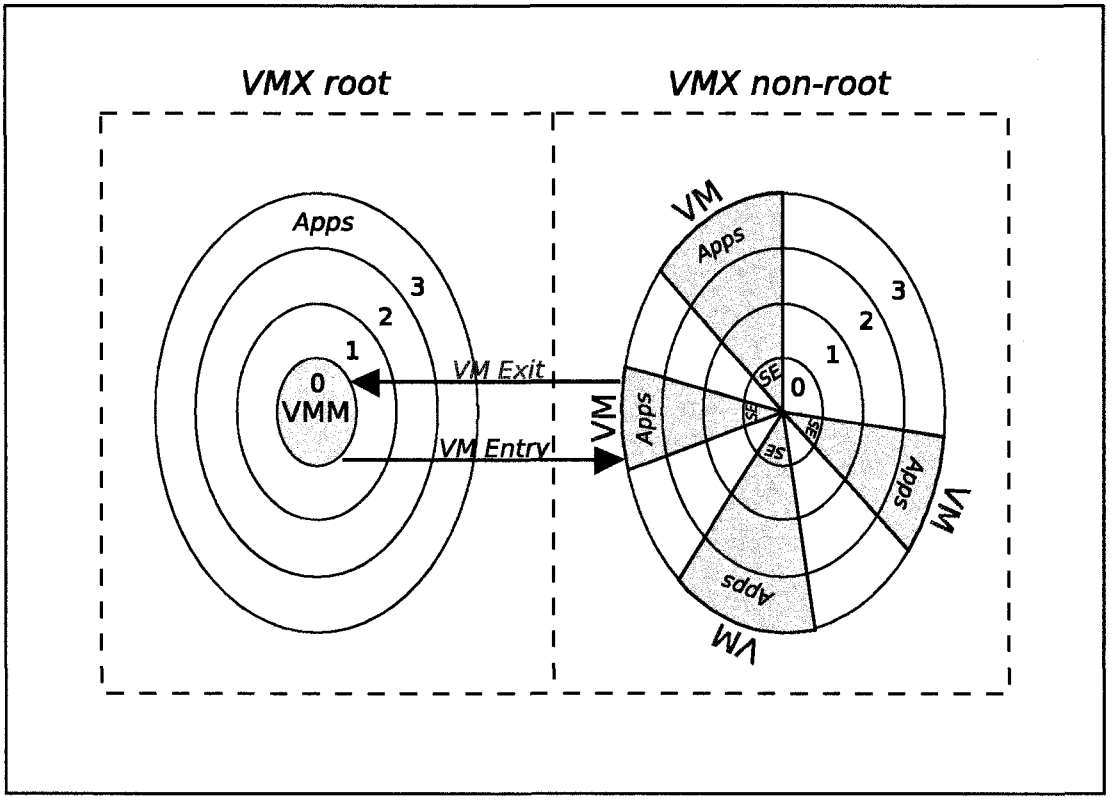

FIG. 1.5: Les processeurs x86 avec la technologie Intel-VT ou AMD-V supportent deux niveaux d'opération, chacun composé par quatre modes d'opération

nées dans la mémoire. Cette action est exécutée par le système d'exploitation à l'aide de l'instruction sensible MOV du ISA x86. Pourtant, comme le système d'exploitation de la machine virtuelle est lui aussi exécuté dans le niveau d'opération invité, ne possèdant pas les privilèges nécessaires pour exécuter une tel instruction, la tentative d'exécution de cette instruction déclenchera une *VM Exit*, le contrôle retournant au VMM.

Les transitions sont gérées par une nouvelle structure de contrôle, baptisée VMCS (de l'anglais *Virtual Machine Control Structure*) chez Intel et VMCB (de l'anglais *Virtual Machine Control Block*) chez AMD, qui garde une trace de l'événement ou de l'instruction qui a déclenché la transition (pour l'exemple précédent, celle-ci sera l'instruction MOV exécutée par le système d'exploitation invité). La VMCS est divisée logiquement en deux sections (ou tableaux), une pour le niveau d'opération hôte et une autre pour le niveau d'opération invité. Ces sections contiennent des champs pour chacun des registres qui définissent l'état du processeur [43]. De cette manière, quand une instruction *VM entry* est exécutée et que le contrôle passe à une machine virtuelle, le contenu de la section réservée au niveau d'opération invité est chargé à l'aide de l'instruction *VMCS read*. Ensuite, quand le processeur revient au niveau d'opération hôte (et au contrôle du VMM) avec l'exécution de l'instruction *VM exit*, l'état du processeur « virtuel » est enregistré dans la section réservée au niveau d'opération invité à l'aide de l'instruction *VMCS write* et le contenu de la section réservée au niveau d'opération hôte est rechargé.

Dans le contexte de la virtualisation, le fait que la VMCS enregistre séparément l'état du processeur pour chacun des niveaux d'opération simplifie le fonctionnement du VMM car sinon le VMM devrait se charger lui-même de ce contrôle, qui devrait alors être implementé par logiciel.

L'approche utilisée par Intel et AMD pour rendre leurs nouvelles générations de processeurs « virtualisables » est assez particulière. Au lieu de réarranger l'ensemble des instructions qui composent l'ISA de cette architecture et corriger le comportement des 17 instructions sensibles et non privilégiées, ils ont créé un nouveau niveau d'opération du processeur conçu spécialement pour l'exécution des machines virtuelles. Une des raisons de ce choix, qui constitue depuis longtemps une des principales priorités de Intel et de AMD, est de garder les nouveaux processeurs rétro-compatibles avec les

anciennes générations de processeurs. Cette caractéristique permet aux applications écrites pour les anciennes générations de fonctionner avec les nouveaux processeurs. En pratique, si l'ordinateur n'est pas virtualisé (s'il n'y a pas de machines virtuelles en exécution utilisant les extensions AMD-V/Intel-VT), le processeur fonctionnera en permanence dans le mode d'opération hôte.

La première génération des processeurs AMD-V et Intel-VT se limite au support matériel pour la virtualisation du processeur. Pourtant, il s'agit de technologies encore en développement. Lamelot [34] rapporte que le nombre de cycles de calcul utilisé par les instructions *VM entry*, *VM exit*, *VMCS read* et *VMCS write* du processeur Intel Pentium 4 est en moyenne deux fois plus grand que celui de son successeur, le processeur Intel Core 2 Duo.

1.3.1.2 Virtualisation de la mémoire

La correspondance entre la mémoire virtuelle et la mémoire physique de l'ordinateur est réalisée par une unité de l'UCT appelée MMU (de l'anglais *Memory Management Unit*, ou unité de gestion de la mémoire) [34]. La MMU utilise un tableau pour faire cette correspondance, appelé PT (de l'anglais *Page Table*, ou table de pages). Les systèmes d'exploitation invités ne doivent pas pouvoir modifier ce tableau directement, sous peine de compromettre le fonctionnement de la machine. Le VMM doit intervenir à chaque fois que les systèmes d'exploitation invités ont besoin d'accéder à la mémoire et modifier les tableaux de correspondance lui-même.

Pour assurer l'isolation des machines virtuelles, le VMM utilise une table de pages pour chaque machine virtuelle, appelée SPT (de l'anglais *Shadow Page Table*, ou table de pages « fantôme »). Celles-ci font la correspondance entre les adresses mémoire réclamées par les systèmes d'exploitation invités et les adresses de la mémoire physique réservées en avance à chaque machine virtuelle par le VMM [34]. Cette approche, illustrée par la figure 1.6, impose une surcharge considérable au fonctionnement du système, car le VMM doit intervenir et exécuter des opérations complexes chaque fois qu'une machine virtuelle a besoin d'accéder à la mémoire et ce même s'il s'agit d'une plage d'adresses réservée à son utilisation.

Pour diminuer le travail du VMM et améliorer la performance du système, la deuxième génération de processeurs AMD-V et Intel-VT inclut le support matériel à

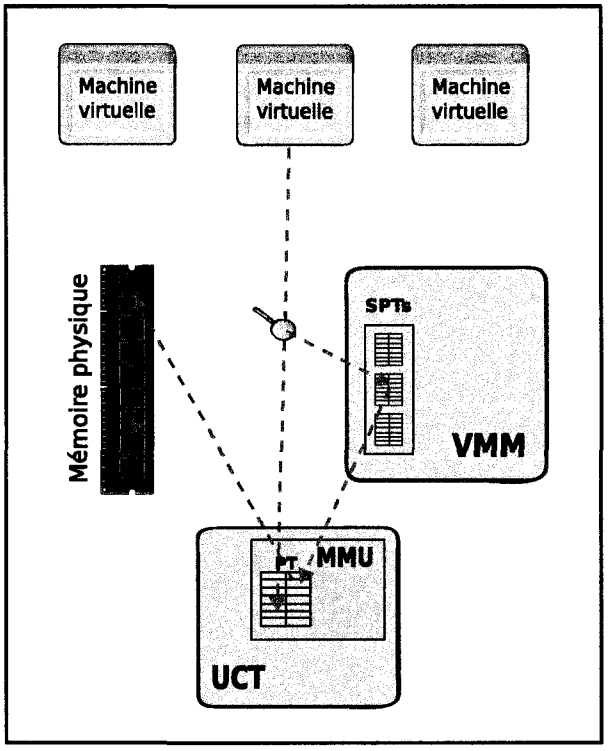

FIG. 1.6: Accès à la mémoire avec des tables de correspondance d'adresses implantées par logiciel (SPTs).

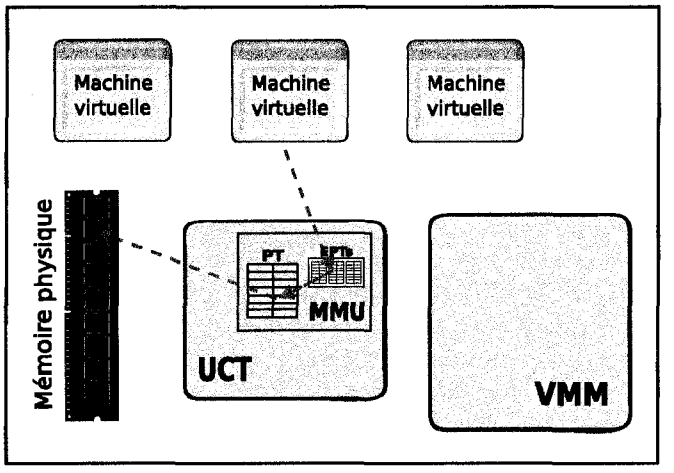

FIG. 1.7: Accès à la mémoire avec des tables de correspondance d'adresses implantées par matériel (EPTs).

la virtualisation de la mémoire. Il s'agit de l'implantation des SPT directement sur le processeur, lesquels sont appelées NPP (de l'anglais *Nested Page Tables*, ou tables de pages imbriquées) chez AMD et EPT (de l'anglais *Extended Page Tables*, ou tables de pages étendues) chez Intel. Ce mécanisme permet aux systèmes d'exploitation invités d'accéder aux plages d'adresses de la mémoire qui leur sont réservées sans la médiation du VMM. La figure 1.7 illustre l'utilisation de ce nouveau mécanisme. La machine virtuelle au centre de la figure accède librement à la plage d'adresses qui lui est allouée par le VMM au moment de son initialisation et enregistrée dans la structure EPT. La traduction des « adresses virtuelles » (EPT) en adresses physiques de la mémoire (PT) est faite par le matériel et ne demande pas l'intervention du VMM.

1.3.1.3 Virtualisation des périphériques

Présentement, les fabricants de processeurs concentrent leur effort dans le développement du support matériel pour la virtualisation des périphériques. Pour l'instant, un VMM conçu pour les processeurs AMD-V et Intel-VT présente aux machines virtuelles des périphériques génériques virtuels émulés. Toutes les requêtes faites par les machines virtuelles aux périphériques passent par le VMM qui les redirige vers le matériel [34].

La virtualisation des périphériques permettrait aux machines virtuelles de com-

muniquer directement avec un périphérique particulier (dans le cas où l'utilisation de ce périphérique est allouée à la machine virtuelle par le VMM) ou, au moins, de communiquer avec une instance virtuelle de ce périphérique, ce qui permettrait l'utilisation des pilotes spécifiques conçus et optimisés par le fabricant. Cette approche doit améliorer la performance des opérations d'entrée-sortie exécutées par les machines virtuelles et rendre plus léger le rôle du VMM.

1.4 Avantages et cas d'utilisation de la virtualisation

La virtualisation, en plus de faciliter la consolidation des serveurs, présente une série d'autres avantages qui méritent d'être mentionnés. Quelques-uns, comme la migration en cours de fonctionnement, sont propres à la virtualisation et pourraient difficilement être implantés dans un autre environnement. Cette section fait un survol des principaux avantages et des principaux cas d'utilisation de la virtualisation.

Réduction des coûts d'entretien et de matériel

La consolidation du nombre de serveurs à l'aide de la virtualisation affecte les coûts d'acquisition et d'entrétien du matériel du parc informatique de différentes manières.

1. Économie des ressources environnementales. La consommation électrique et les besoins de refroidissement et de ventilation d'un serveur sous-utilisé sont presque identiques à ceux d'un serveur utilisé à 80% de sa capacité [3]. Il est généralement admis [3] que pour chaque dépense d'un dollar en utilisation d'équipement informatique correspond une dépense équivalente en refroidissement. Une réduction du nombre de serveurs utilisés mène à une économie des ressources environnementales.
2. Réduction de l'espace requis dans la salle des machines. Un nombre réduit de serveurs occupent moins d'espace dans la salle de machines. Cet avantage est particulièrement important pour les entreprises qui possèdent une petite salle des machines et favorise l'expansion future de leur parc informatique.
3. Économie d'équipements informatiques auxiliaires. Un nombre réduit de serveurs requiert l'utilisation de moins d'équipements informatiques auxiliaires,

tels que les commutateurs réseau, les commutateurs écran-clavier-souris, les *UPS* (système d'alimentation sans coupure) et même le câblage réseau.

4. Économie sur l'entretien physique des machines. Il est plus facile et moins coûteux d'exécuter les tâches d'entretien physique de 50 serveurs (avec 10 serveurs virtuels chacun) que celles de 500 serveurs.

Support des applications et des systèmes d'exploitation patrimoniaux

Par application patrimonial (de l'anglais *legacy application*) on fait référence aux applications logicielles conçues pour être exécutées dans des systèmes d'exploitation hérités (de l'anglais *legacy OS*), lesquels dépendent de bibliothèques de programmes incompatibles avec les systèmes d'exploitation actuels ou, encore, d'une classe de matériel informatique qui n'est plus dans la norme d'aujourd'hui. Quelques-unes de ces applications ont été faites sur mesure pour répondre à des besoins spécifiques et il n'est simplement pas possible, trop coûteux ou trop risqué de les rendre compatibles avec les plateformes actuelles.

La virtualisation peut être une solution intéressante pour le support d'applications patrimoniaux. Si pour exécuter une certaine application il faut utiliser un système d'exploitation hérité, il est possible d'installer ce système d'exploitation dans une machine virtuelle, plutôt que de lui dédier un ordinateur au complet. De plus, les fabricants de matériel informatique n'offrent plus de support aux systèmes d'exploitation hérités. Cela rend difficile, par exemple, de faire fonctionner une carte vidéo récente sur un système d'exploitation comme *Windows 95* - il n'existe simplement pas de pilote disponible pour ce système d'exploitation. Pourtant, certains logiciels de virtualisation présentent aux machines virtuelles une abstraction du matériel physique, utilisant généralement du matériel générique qui ne nécessite pas des pilotes spécifiques pour fonctionner.

Finalement, la situation où deux applications distinctes demandent le support de deux bibliothèques incompatibles peut être résolue avec la virtualisation - il suffit d'installer chaque application dans une machine virtuelle différente.

Environnement sécuritaire pour l'exécution d'applications

La virtualisation permet la création d'un environnement sécuritaire pour l'exécution d'applications : les machines virtuelles. La sécurité vient du mécanisme d'isolation imposé par la couche de virtualisation, ce qui rend les machines virtuelles isolées les unes des autres, ainsi que du système hôte qui les héberge. Ainsi, en théorie, un virus contaminant une machine virtuelle n'affectera pas le fonctionnement des autres [34]. De manière similaire, la machine virtuelle constitue un environnement idéal pour l'exécution d'applications inconnues (auxquelles on ne fait pas confiance) car celles-ci ne peuvent endommager que la machine virtuelle sur laquelle elles sont exécutées [58].

En pratique, des failles de sécurité ont déjà été détectées dans certaines solutions de virtualisation [45]. Si les mécanismes d'isolation ne sont pas bien conçus, la sécurité peut être compromise [69]. Si ces mécanismes sont bien conçus, cela ajoute une barrière de plus à franchir.

Environnement idéal pour le développement, les tests et la mise au point d'applications

La virtualisation constitue une bonne solution pour partager un environnement de développement entre plusieurs développeurs. En dédiant une machine virtuelle à chaque développeur on assure que le travail exécuté par l'un n'affectera pas celui de l'autre.

Certaines technologies de virtualisation permettent l'exécution de plusieurs systèmes d'exploitation différents sur un même ordinateur, un par machine virtuelle. Cette caractéristique enlève le besoin d'avoir une infrastructure de tests composée de plusieurs ordinateurs qui sont utilisés simplement pour tester le code sur différents systèmes d'exploitation. Par exemple, il est commun pour des compagnies faisant du développement web de vérifier si leurs projets fonctionnent avec différents navigateurs tels qu'Internet Explorer et Firefox. Avec la virtualisation, il est possible d'avoir plusieurs machines virtuelles fonctionnant sur un même ordinateur, chacune configurée avec un environnement de tests différent.

Gestion centralisée

La virtualisation offre des mécanismes qui facilitent la centralisation de la gestion du parc informatique. Ceci est fait à l'aide d'interfaces de gestion puissantes qui permettent non seulement de surveiller l'utilisation des ressources des serveurs virtualisés et des machines virtuelles mais aussi de modifier dynamiquement la distribution de la charge de travail entre les serveurs qui composent le parc informatique. Les interfaces de gestion permettent aussi de démarrer des activités de manutention du parc informatique, comme la sauvegarde, la migration et la relève rapide de machines virtuelles, l'objectif étant d'optimiser l'utilisation des ressources disponibles.

Sauvegarde facilitée

Il est plus facile de sauvegarder un serveur virtuel qu'un serveur physique.

Normalement, la procédure utilisée pour la sauvegarde d'un serveur physique fait la séparation des fichiers du système en trois groupes : les fichiers des logiciels, les fichiers de configuration et les données brutes résultantes de l'utilisation des services offerts par le serveur, les deux derniers groupes étant effectivement sauvegardés. Chaque nouveau serveur ajouté au parc doit être scrupuleusement configuré dans le système de sauvegarde, une tâche qui demande beaucoup de temps pour un administrateur et qui finit par retarder le démarrage de la sauvegarde.

Dans le cas d'un serveur virtuel, la procédure est plus simple : on sauvegarde la machine virtuelle au complet. Le format et la taille des machines virtuelles varient selon la technologie de virtualisation utilisée, certaines étant plus faciles à sauvegarder que d'autres. Il existe des produits intégrant la sauvegarde aux outils de gestion, ce qui rend la tâche (semi) automatique et donc possiblement plus facile à gérer.

Migration en cours de fonctionnement

La migration en cours de fonctionnement (de l'anglais *« live migration »*) est une technique offerte par certaines technologies de virtualisation qui permet de « geler » l'état de fonctionnement d'une machine virtuelle (l'ensemble des processus qui s'exécutent sur le système d'exploitation, le contenu de la mémoire, etc.) de façon à pouvoir

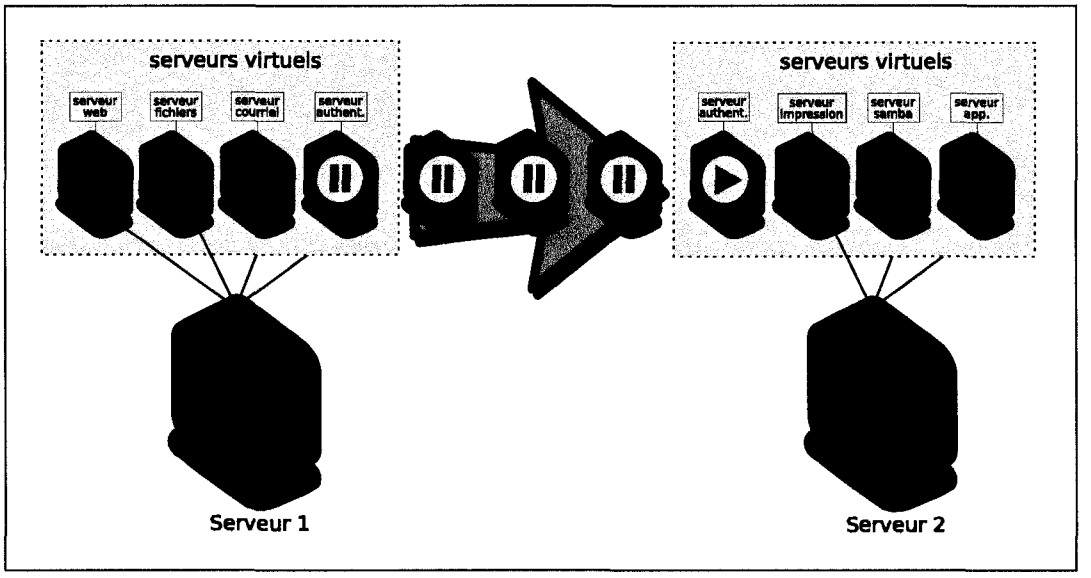

FIG. 1.8: Migration en cours de fonctionnement d'un serveur virtuel

faire le transfert, ou la migration, de la machine virtuelle d'un serveur physique à un autre par le réseau. Une fois le transfert terminé, la machine est remise en opération. À part le temps nécessaire pour compléter ce transfert, qui peut se limiter à quelques secondes seulement, cette procédure, illustrée par la figure 1.8, est généralement transparente pour les utilisateurs des services offerts par la machine virtuelle. Il n'y a pas d'arrêt temporaire de service, mais la migration en cours de fonctionnement peut entraîner la perte temporaire de la connexion réseau. Habituellement, ceci ne constitue pas un problème car la majorité des applications offrant des services utilisent des paquets TCP, lesquels sont renvoyés si le destinataire ne confirme pas leur reception.

Cette technique permet de mieux répartir la charge (l'utilisation du temps de l'UCT et de la mémoire, par exemple) sur l'ensemble de machines du parc informatique [29]. Par exemple, si le taux d'utilisation d'un serveur dépasse une certaine limite, le système peut être configuré pour migrer une des machines virtuelles vers un serveur moins utilisé.

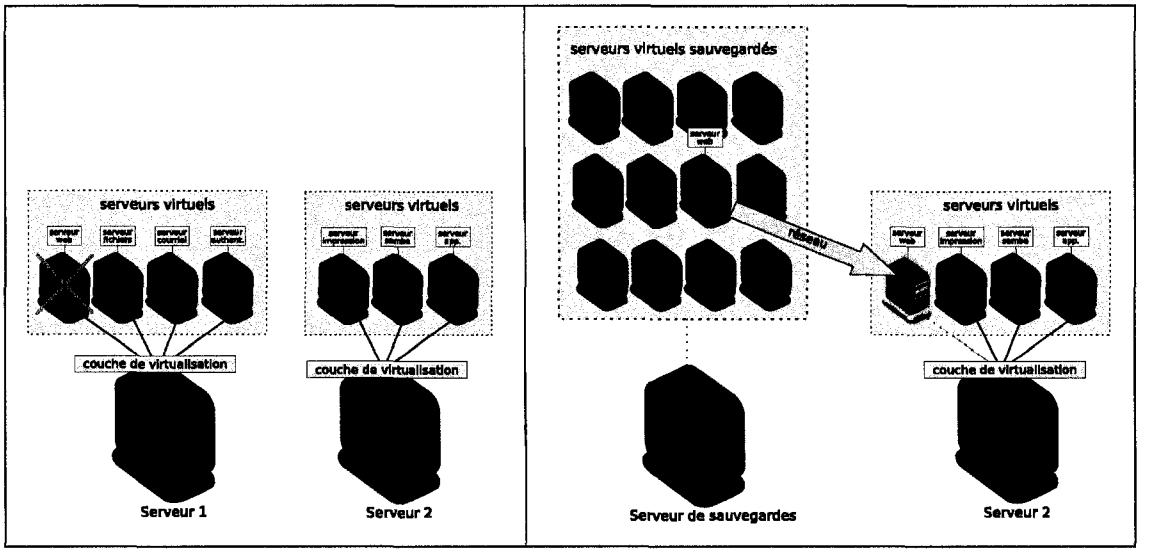

FIG. 1.9: Remise en service d'un serveur virtuel à partir d'une copie de sauvegarde

Relève rapide

Même les technologies de virtualisation qui n'offrent pas la migration en cours de fonctionnement permettent de profiter de la flexibilité offerte par la relève rapide des machines virtuelles. La relève consiste en la remise en fonction d'un serveur à partir d'une copie de sauvegarde. Comme les machines virtuelles peuvent fonctionner sur n'importe quel serveur contenant la couche de virtualisation appropriée, la virtualisation permet une remise en service de l'infrastructure beaucoup plus rapide, le temps de mise en service d'une machine virtuelle étant directement relié au transfert de la dernière copie de sauvegarde du serveur de sauvegarde au serveur qui prend la relève, comme l'illustre la figure 1.9.

Haute disponibilité

La haute disponibilité d'un service est la capacité d'offrir ce service de façon continue même si la machine responsable de l'offrir tombe en panne. On atteint la haute disponibilité d'un service avec la redondance des installations.

Cette technique était auparavant réservée uniquement aux services critiques, en raison de son coût élevé. Par exemple, dans les années 1980, la compagnie *Tandem*

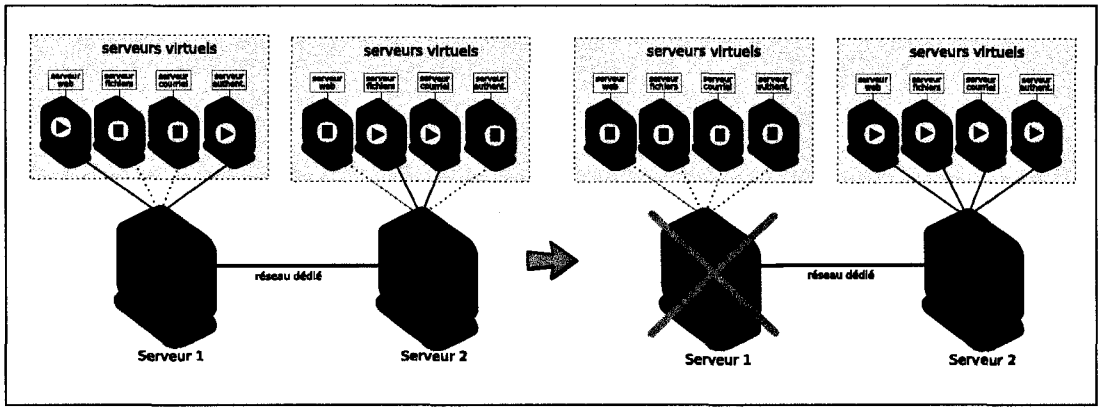

FIG. 1.10: Une infrastructure virtualisée hautement disponible

Computer Systems était connue pour produire des ordinateurs tolérants aux fautes avec l'architecture *NonStop* [21], laquelle utilisait la redondance matérielle et logicielle.

Aujourd'hui, avec l'utilisation de technologies comme *Heartbeat* et *DRBD* [49], il est possible de rendre une application hautement disponible en utilisant deux ordinateurs connectés par un réseau dédié. Cette architecture utilise un schéma où un ordinateur fonctionne comme « maître » et l'autre comme « esclave ». Le logiciel DRBD est responsable de synchroniser le disque de l'ordinateur esclave avec le contenu du disque de l'ordinateur maître, utilisant pour cette activité le réseau dédié qui connecte les deux ordinateurs ensemble. Le logiciel Heartbeat, installé sur les deux machines, surveille leur fonctionnement. S'il detecte par moyen d'une série de tests de connexion que l'ordinateur maître ne fonctionne plus correctement, l'ordinateur esclave prend la rèleve. Le niveau de disponibilité offert par l'architecture Heartbeat+DRBD, même s'il n'est pas du même niveau que celui offert par l'architecture *NonStop*, est suffisant pour un très grand nombre d'applications.

La virtualisation permet, au prix d'un investissement minimal, l'utilisation de techniques de haute disponibilité tel Heartbeat+DRBD sur des serveurs virtualisés. Une installation utilisant uniquement une paire de machines permet de rendre plusieurs serveurs virtuels hautement disponibles, comme l'illustre la figure 1.10.

Distribution d'applications sous la forme de « serveurs mono-fonctionnels virtuels »

La virtualisation a donné une nouvelle signification au terme *serveur mono-fonctionnel* (un serveur dédié qui offre un seul service, comme un serveur d'impression). Une machine virtuelle peut être personnalisée pour l'exécution d'un logiciel particulier. Le système d'exploitation peut être simplifié de façon à ne contenir que le strict nécessaire au fonctionnement de ce logiciel (par exemple, le noyau d'un serveur web n'a pas besoin de charger un module *bluetooth*). Une telle machine virtuelle personnalisée pour le fonctionnement d'une application spécifique est appelée un serveur mono-fonctionnel virtuel (de l'anglais *virtual appliance*).

De la même façon qu'un logiciel est empaqueté dans un format normalisé (comme *.msi*, *.rpm*, *.deb*), il est aussi possible d'empaqueter le serveur mono-fonctionnel au complet. Pour les utilisateurs, le principal avantage d'obtenir un logiciel dans ce format est qu'il n'a pas besoin de vérifier si les dépendances du logiciel sont satisfaites. L'environnement d'exécution est déjà prêt à être utilisé : il suffit d'installer et de démarrer la machine virtuelle. Par exemple, la compagnie *VMware*, propriétaire du logiciel de virtualisation *VMware Server,* offre sur son site web le téléchargement de plusieurs serveurs mono-fonctionnels dans le format *Open Virtual Machine Format* (.ovf).

Pour les développeurs, cette approche facilite le support du produit, car plutôt que de prévoir plusieurs scénarios différents de déploiement, ils peuvent se concentrer sur un seul. Pourtant, il n'existe pas aujourd'hui de format normalisé de machine virtuelle. Les différentes technologies de virtualisation n'étant pas toujours compatibles entre elles, le choix de la technologie (ou format) à utiliser varie.

Gestion contrôlée des ressources

Un des facteurs de risque relié à consolidation de plusieurs applications sur un même serveur est le manque de contrôle sur les ressources utilisées par chaque application [33]. Il est fort possible dans cette situation qu'une application monopolise de façon abusive certaines ressources.

La virtualisation permet la création d'environnements d'exécution avec des res-

sources limitées, le but étant d'augmenter la qualité de service en garantissant une portion spécifique des ressources physiques du serveur aux différentes machines virtuelles qu'il héberge [42]. Le succès de cette approche dépend de l'utilisation de bons ordonnanceurs [58] par les technologies de virtualisation.

Le contrôle des ressources est une caractéristique qui contribue à diminuer le risque associé à la consolidation des serveurs, car elle évite qu'une application monopolise, par exemple, l'UCT, limitant le temps de calcul du serveur à une seule machine virtuelle.

Chapitre 2

Révision des technologies de virtualisation

De nos jours, l'implantation de la virtualisation telle que décrite par Popek & Goldberg (voir section 1.2.1), dans laquelle le VMM intercepte et interprète les instructions sensibles exécutées par les machines virtuelles (connue en anglais par l'expression « *trap and emulate* »), n'est pas la seule méthode utilisée pour la virtualisation de serveurs. Au cours des années, divers logiciels de virtualisation ont été développés utilisant d'autres technologies. Ces technologies diffèrent dans le degré d'abstraction qu'elles font du matériel sous-jacent ainsi que dans la manière dont la couche de virtualisation est conçue [51]. Plusieurs études [42, 58, 29, 34, 32, 35, 30, 59, 24, 12] ont été consacrés à la révision de ces technologies.

Ce chapitre présente une révision des principales technologies utilisées actuellement pour la virtualisation des serveurs sur l'architecture x86. Il est commun de rencontrer dans la littérature l'utilisation de différentes nomenclatures pour désigner la même technologie de virtualisation [29]. Nous allons utiliser celles qui sont les plus couramment employées. Une analyse qualitative, basée sur les études mentionnées ci-haut, comparant la flexibilité ainsi que la performance des technologies de virtualisation, est présentée à la fin du chapitre.

2.1 Virtualisation complète

La technique connue par le nom de virtualisation complète (de l'anglais *full virtualization*) consiste à émuler par logiciel, de manière partielle ou encore intégrale, le matériel qui compose la machine physique et de présenter cette version « virtuelle » du matériel au système d'exploitation invité [7]. Aucune modification à ce dernier n'est requise car il n'est pas informé du fait qu'il s'exécute sur une machine virtuelle.

La base des logiciels qui emploient la virtualisation complète est l'émulation matérielle. Mais l'émulation toute seule n'équivaut pas à la virtualisation. Il existe présentement deux techniques qui sont utilisées avec l'émulation du matériel pour implanter la virtualisation complète : la *traduction binaire* des instructions et, quand les caractéristiques du processeur le permettent, la technique classique, ou *virtualisation classique*, comme est appelée aujourd'hui la technique décrite par Popek & Goldberg [50].

2.1.1 L'émulation matérielle

L'émulation matérielle consiste à émuler l'intégralité des composants qui forment une machine. La base de l'émulation matérielle est l'émulation du processeur, plus précisément du jeu d'instructions élémentaires de l'architecture (voir section 1.2.2), par logiciel [42]. Le logiciel responsable d'exécuter cette tâche est appelé *émulateur*. Les émulateurs n'ont pas comme objectif primaire la virtualisation. Ils ont été conçus pour reproduire le comportement d'une architecture différente de l'architecture native [58]. Par exemple, avec un émulateur de l'architecture PowerPC, il est possible d'exécuter une application compilée pour cette plateforme dans un ordinateur équipé avec un processeur x86. Pour ce faire, chacune des instructions envoyées au processeur PowerPC émulé doit être convertie en instructions x86 puis exécutée par le processeur physique. On dit dans ce cas que les instructions sont interprétées ou émulées.

Quand on fait l'émulation d'une architecture identique à l'architecture native, les instructions envoyées au processeur émulé n'ont pas nécessairement besoin d'être interprétées par l'émulateur avant d'être exécutées. En effet, comme celles-ci appartiennent au jeu d'instructions élémentaires (l'ISA) du processeur réel, elles peuvent, pour cette raison, être exécutées directement par ce dernier. Pourtant, dans le cas de

l'architecture x86, en raison des caractéristiques mentionnées dans la section 1.3, il est tout de même nécessaire pour l'émulateur d'interpréter la totalité des instructions.

La « portabilité » des applications constitue le principal avantage de l'émulation matérielle. Deux exemples de logiciels qui font de l'émulation matérielle sont *Bochs* [35] et *QEMU* [4]. La spécialité de Bochs est l'émulation de toute la série x86 d'Intel, du 386 jusqu'au x86-64. QEMU est capable d'émuler diverses plateformes en plus de l'architecture x86, telles que SPARC, MIPS, PowerPC et ARM. Le principal inconvénient des émulateurs est leur performance. Comme chaque instruction exécutée par le système émulé doit être interprétée par le processeur réel, les pénalités en temps d'exécution sont significatives [42]. Jones [29] mentionne que les ralentissements peuvent être de l'ordre de 100 ou même de 1000 pour certaines configurations spécifiques. Dans la limite extrême de ce contexte, une instruction qui prend normalement 2 milisecondes pour s'exécutér dans le système natif peut prend jusqu'à 2 secondes pour s'exécuter dans le système émulé.

QEMU est un exemple de logiciel qui peut fonctionner soit comme un « émulateur pur » soit comme une technologie de virtualisation pour l'architecture x86. Dans le deuxième cas, il utilise un pilote spécial, ou « accélérateur », appelé *KQEMU*[1] [5], ainsi que la technique de traduction binaire [28], présentée dans la section suivante.

2.1.2 Traduction binaire

La traduction binaire des instructions, ou simplement traduction binaire (de l'anglais *binary translation*), est une approche développée pour contourner les caractéristiques de certains processeurs, comme ceux de l'architecture x86, qui les empêche d'être virtualisés de la manière classique. En effet, comme l'exécution de certaines instructions sensibles par des processus non privilégiés ne génère aucune exception sur ce type de processeur, il fallait trouver une technique qui permettrait de contourner ce problème. La solution employée par la technique de traduction binaire consiste à surveiller le code exécuté par le système d'exploitation invité et à modifier (ou tra-

[1] KQEMU est considéré comme un « accélérateur » car, comme les instructions d'une machine virtuelle KQEMU font partie de l'ISA du processeur physique et qu'elles ne nécessitent aucune traduction avant d'être exécutées, la performance du logiciel (QEMU) est augmentée (du moins en théorie) considérablement.

duire) les séquences d'instructions qui peuvent changer l'état du système hôte (les instructions sensibles) par d'autres instructions qui produisent le même effet. Les autres instructions peuvent être exécutées directement par le processeur.

L'utilisation de cette approche sans l'assistance d'un mécanisme d'optimisation impose une perte d'efficience significative à l'exécution des machines virtuelles. Pour améliorer leur performance, les logiciels de virtualisation qui utilisent cette approche, comme *VMware ESX Server* [72], *Microsoft Virtual PC* [42] et KQEMU, font aussi l'emploi d'une cache de traduction (de l'anglais *translation cache*) pour enregistrer les instructions récemment traduites pour l'utilisation future ainsi que de plusieurs autres stratégies, comme le *Patch Manager* utilisé par VirtualBox [19], qui plutôt que de continuellement traduire les instructions sensibles remplace certaines parties récurrentes du code faisant appel à ce type d'instruction par du code faisant appel à d'autres instructions (inoffensives) mais qui produisent les mêmes effets.

Une différence essentielle entre l'approche que nous venons de décrire et celle employée par les émulateurs purs est que, dans le cas des émulateurs, la totalité des instructions sont interprétées tandis que dans le cas de la traduction binaire avec l'emploi d'une cache, une instruction qui a déjà été interprétée ne le sera plus lors de ses exécutions subséquentes.

2.1.3 Virtualisation classique

Pour qu'une architecture puisse être virtualisée de la manière classique, comme décrit par Popek & Goldberg [50], les instructions sensibles qui composent le jeu d'instructions du processeur ne doivent pas pouvoir être exécutées par une application non privilégiée sans générer une faute ou une exception. Dans cette technique de virtualisation, les systèmes d'exploitation invités sont exécutés en tant qu'applications non privilégiées et chaque fois qu'ils exécutent une de ces instructions le VMM l'intercepte et l'interprète. Cette technique a été décrite en détails dans la section 1.2.3.

Comme indiqué à la section 1.3, cette technique ne peut pas être utilisée par des machines utilisant des processeurs x86 standards. Une partie des instructions sensibles de l'ISA de cette architecture ne génèrent pas une exception quand elles sont exécutées par une application non privilégiée. Cette caractéristique empêche l'implantation du

VMM décrit par Popek & Goldberg pour cette architecture.

Récemment, Intel et AMD ont intégré des extensions supplémentaires au ISA x86 de leurs nouvelles générations de processeurs avec l'objectif de faciliter la virtualisation de cette architecture. Les détails concernant ces changements ont été présentés dans la section 1.3.1. De manière simplifiée, un nouveau niveau d'opération a été ajouté au processeur. Pour qu'un logiciel de virtualisation puisse exploiter ces nouvelles extensions il doit exécuter les machines virtuelles dans ce nouveau niveau d'opération du processeur, ceci étant moins privilégié que le niveau d'opération *standard* dans lequel le VMM est exécuté. Toute tentative d'exécution d'une instruction sensible par une machine virtuelle entraîne automatiquement un retour au mode d'opération standard. En pratique, cela a le même effet que la génération d'une exception et donne au VMM l'opportunité d'intercepter et d'interpréter l'instruction en question.

Un exemple d'un logiciel de virtualisation classique pour l'architecture x86 est *KVM* (de l'anglais *Kernel-based Virtual Machine*, ou machine virtuelle basée sur le noyau) [31]. KVM est composé de deux parties : un sous-système, ou module, incorporé au noyau de Linux et un programme utilisateur. Le module noyau exploite les nouvelles extensions de l'architecture x86 pour transformer Linux en un VMM. Le programme utilisateur est une version modifiée d'une application QEMU, adaptée pour que toutes les instructions sensibles exécutées par le système d'exploitation invité fassent appel au module noyau [7].

De manière générale, la virtualisation conçue avec des techniques qui exploitent ces nouvelles extensions est aussi appelée *virtualisation assistée par le matériel* (de l'anglais *hardware-assisted virtualization*) [41]. L'utilisation de ces extensions supplémentaires n'est pas exclusive aux logiciels qui emploient la virtualisation classique. VirtualBox utilise (de manière optionnelle) une partie de ces extensions (et pas de la manière prévue par Intel [19]) pour la conception de son VMM et Xen les emploie quand il doit exécuter un système d'exploitation non modifié (comme Windows XP) dans une machine virtuelle.

2.2 Paravirtualisation

La technique connue sous le nom de paravirtualisation est une alternative à la virtualisation complète. Une partie considérable de la complexité inhérente à la virtualisation des serveurs est reliée au fait que les systèmes d'exploitation sont conçus pour avoir un accès exclusif à l'ordinateur. Comme ils ne sont pas conçus pour partager le contrôle du matériel avec une autre application, ils ont la liberté de l'utiliser de la manière qui leur convient le mieux. Quand un ordinateur est virtualisé avec une technique de virtualisation complète et que plusieurs systèmes d'exploitation partagent l'utilisation de la machine, c'est le VMM qui s'occupe de gérer les tentatives d'accès concurrents au matériel tout en donnant l'illusion aux systèmes d'exploitation invités qu'ils sont les seuls à contrôler la machine. Cette approche fonctionne très bien mais demande au VMM beaucoup d'effort pour coordonner l'exécution des machines virtuelles.

Il est possible d'alléger le travail du VMM en informant les systèmes d'exploitation invités qu'ils n'ont pas un accès exclusif au matériel et qu'ils doivent le partager entre eux [15]. Pour coordonner le partage de manière efficace, le VMM offre aux machines virtuelles une interface spéciale qui doit être utilisée par les systèmes d'exploitation invités pour accéder au matériel.

La clé du succès (performance) de cette technologie de virtualisation est basée sur la collaboration des systèmes d'exploitation invités qui sont informés qu'ils s'exécutent à l'intérieur d'une machine virtuelle. Pour ce faire, les systèmes d'exploitation invités doivent être modifiés, de manière à pouvoir utiliser cette interface spéciale offerte par le VMM qui dans ce contexte est appelé hyperviseur [59].

De manière générale, une machine virtuelle exécutée dans un système paravirtualisé présente une bien meilleure performance que celle d'une machine virtuelle exécutée avec une technique de virtualisation complète [15]. D'après Bonnet [7], les deux goulots d'étranglement d'un système virtualisé sont la gestion de la mémoire et la gestion des opérations d'entrée/sortie (E/S). Une application s'exécutant sur une machine virtuelle utilisant une technique de virtualisation complète et qui désire accéder à l'Internet, par exemple, doit traverser plusieurs couches d'abstraction virtuelles (le matériel émulé par le VMM). La même application s'exécutant sur un système pa-

ravirtualisé utilisera des pilotes spéciaux pour contourner ces couches d'abstraction et communiquer directement avec l'interface réseau de la machine physique (ce type d'accès étant tout de même toujours contrôlé par le VMM).

La principal désavantage de la paravirtualisation vient justement du fait que les systèmes d'exploitation invités doivent être modifiés pour mieux collaborer avec le VMM. Les changements visent à adapter toutes les couches qui accèdent au matériel afin qu'elles puissent utiliser l'interface spéciale offerte par le VMM [7]. De plus, cela limite l'utilisation de la paravirtualisation aux systèmes d'exploitation libres (pour lesquels le code source est disponible pour que les changements puissent être appliqués) ou, encore, aux systèmes d'exploitation propriétaires offerts avec les modifications nécessaires à cet effet.

Deux exemples de logiciels de virtualisation qui emploient la technique de paravirtualisation sont *Xen* [2] et *Denali* [68].

2.3 Virtualisation au niveau du système d'exploitation

La virtualisation au niveau du système d'exploitation (de l'anglais *Operating System (OS) level virtualization*) est une technique qui implémente la virtualisation d'une manière très différente de celles présentées jusqu'à maintenant. Une caractéristique commune à toutes les techniques de virtualisation décrites jusqu'à maintenant est l'émulation par logiciel d'au moins un sous-ensemble des composantes matérielles de la machine physique. Cette émulation est nécessaire pour faire cohabiter plusieurs systèmes d'exploitation différents sur la même machine. Pourtant elle est en même temps responsable d'une partie de la perte d'efficience présentée par ces systèmes, ce qui affecte la performance des logiciels de virtualisation qui utilisent cette technique.

La technologie de virtualisation au niveau du système d'exploitation n'utilise pas l'émulation et, en conséquence, ne permet pas l'exécution de plusieurs systèmes de exploitation. Au lieu de faire l'abstraction du matériel sous-jacent, l'abstraction se fait sur une couche plus élevée : celle au-dessus du système d'exploitation hôte [42]. Ainsi, les machines virtuelles gérées par un logiciel de virtualisation qui emploie cette

technologie partagent non seulement le matériel mais aussi le système d'exploitation hôte. En réalité, c'est le système d'exploitation qui est virtualisé et non la machine physique. Il est possible de voir cette technique de virtualisation comme une extension du concept de la mémoire virtuelle (ou l'inverse, que la mémoire virtuelle n'est qu'un début de ce qui est possible en virtualisation).

L'absence de systèmes d'exploitation invités (dans le cas où le système d'exploitation hôte est partagé) fait que le terme « machine virtuelle » n'est pas employé dans la littérature pour identifier un environnement virtuel créé par un logiciel de virtualisation au niveau du système d'exploitation. Il est remplacé par d'autres expressions telles que « serveur virtuel privé » (de l'anglais *Virtual Private Server* (VPS)) [61] ou simplement « serveur virtuel », ainsi que « conteneur » (de l'anglais *container*).

La virtualisation au niveau du système d'exploitation est une technique qui s'appuie fortement sur des mécanismes renforçant la séparation des processus s'exécutant sur un même système d'exploitation, l'objectif étant d'isoler un processus ou un groupe de processus dans un conteneur dont il est théoriquement impossible de sortir [7]. Ce conteneur forme la base d'un environnement virtuel offert par cette technologie de virtualisation. Pour cette raison, elle est aussi appelée *cloisonnement* et *contextualisation* [59]. Il s'agit d'une technique légère, beaucoup moins complexe à mettre en oeuvre, que permet la création d'un nouveau serveur virtuel rapidement et qui présente une ligère perte de performance. Elle permet le partitionnement d'un serveur physique en plusieurs dizaines de conteneurs presque sans ralentissement.

La majorité des systèmes d'exploitation basés sur UNIX proposent un moyen d'isoler les processus qui peuvent être exploités par les logiciels de virtualisation, comme *chroot* (Linux), *jail* (BSD) et *containers* (Solaris), ainsi que sa version plus évoluée, *zones* (Solaris) [33]. Deux exemples de logiciels qui font l'emploi de la virtualisation au niveau du système d'exploitation Linux de manière similaire sont *Linux-VServer* [14] et *OpenVZ*. Tous les deux sont offerts sous la forme d'une rustine (de l'anglais *patch*) appliqué au noyau de Linux de base, modifiant son mode d'opération standard et fournissant des outils de gestion des environnements virtuels.

Les modifications faites au noyau de Linux par Linux-VServer et OpenVZ incluent, par exemple, la virtualisation du système d'identification des processus (*PID* pour *process identification*) du système d'exploitation avec l'utilisation d'un espace de

nommage (de l'anglais *namespace*) unique par conteneur. Cette modification permet, par exemple, que le PID du processus d'initialisation (*init*) de tous les conteneurs soit 1.

2.4 Analyse qualitative des technologies de virtualisation

Cette section présente une analyse qualitative qui compare la flexibilité ainsi que la performance (théorique) des technologies de virtualisation. Nous ne faisons pas l'analyse de la fiabilité ni de la sécurité de ces technologies. Les résultats de notre étude pratique, présentés dans le chapitre 5, donnent une idée de la fiabilité des logiciels de virtualisation évalués. Par rapport à la sécurité, toutes les technologies de virtualisation présentées dans les sections précédentes répondent à ce critère dans le sens proposé par Popek & Goldberg et présenté dans la section 1.2.1.

Les technologies de virtualisation vues dans ce chapitre ont été présentées en ordre de « flexibilité » décroissante ainsi qu'en ordre de « performance théorique » croissante. Ainsi, un émulateur est un outil très flexible, permettant l'exécution de systèmes d'exploitation conçus potentiellement pour une architecture différente de celle de la machine hôte, mais qui présentent une performance très faible.

Les technologies basées sur la traduction binaire d'instructions sont moins flexibles que les émulateurs car elles exécutent seulement des systèmes d'exploitation compatibles avec l'architecture de la machine hôte. Par contre, elles présentent généralement une performance supérieure à celles des émulateurs. La virtualisation classique profite de l'assistance matérielle à la virtualisation de nouvelles machines pour offrir, en théorie, une meilleure performance.

Les technologies qui utilisent la paravirtualisation ont une flexibilité encore plus restreinte. Leur compatibilité est limitée aux systèmes d'exploitation conçus pour la même architecture que celle de la machine physique et pouvant être modifiés de manière à utiliser l'interface spéciale offerte par l'hyperviseur pour accéder aux composantes matérielles de la machine physique. Cet accès spécial leur confère un gain de performance significatif.

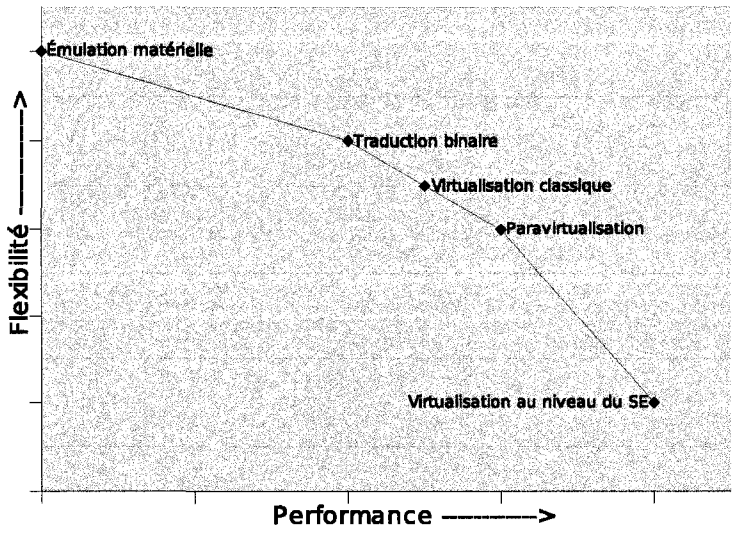

FIG. 2.1: Analyse qualitative de la flexibilité ainsi que de la performance théorique des technologies de virtualisation.

Finalement, les technologies de virtualisation au niveau du système d'exploitation partagent le système d'exploitation hôte avec les serveurs virtuels. Cela limite l'utilisation d'un seul système d'exploitation mais permet aux serveurs virtuels une performance très proche de la performance native (celle expérimentée pour un système n'ayant aucune couche de virtualisation).

L'aspect « flexibilité » des technologies de virtualisation peut être facilement analysé subjectivement en fonction de leur universalité (ou compatibilité) de celles-ci. L'aspect « performance » peut être superficiellement analysé. La pratique peut se montrer très différente de la théorie en ce qui concerne la performance de solutions et pour cette raison nous avons fait un étude pratique comparant la performance des technologies de virtualisation. Le graphique de la figure 2.1 présente une analyse subjective de la flexibilité ainsi que de la performance théorique des technologies de virtualisation.

Il existe au moins une situation où deux des technologies de virtualisation que nous avons présentées sont utilisées conjointement dans le développement d'une so-

lution hybride. Nakajima [41] propose une technologie baptisée *hybrid-virtualization* (ou « virtualisation hybride ») que modifie l'approche utilisée par les logiciels de virtualisation classique (comme KVM), implantée sur les nouveaux processeurs x86 avec support à la virtualisation de manière à profiter de quelques concepts de la paravirtualisation (utilisée par Xen) - comme l'utilisation d'une interface spéciale pour accéder au matériel - mais sans demander la modification des systèmes d'exploitation invités. Il s'agit, cependant, d'un projet expérimental et Nakajima n'indique pas comment son approche améliore la performance de la solution.

Chapitre 3

Révision des travaux similaires

Ce chapitre présente un survol des différentes études publiées au cours des cinq dernières années qui évaluent un logiciel ou un groupe de logiciels qui emploient une des technologies de virtualisation de serveurs présentées dans le chapitre 2. Dans le cas où plus d'un logiciel de virtualisation est évalué par une étude, celle-ci présente aussi une comparaison des résultats obtenus par chacun des logiciels évalués. Nous nous sommes fortement inspiré de ces études dans le choix des métriques et de la méthodologie appliquée lors de notre propre évaluation.

Les caractéristiques principales de ces études sont présentées de manière concise dans un tableau. Ensuite, les différentes charges de travail des tests utilisées par ces études sont brièvement exposées. Le chapitre se termine avec une analyse non exhaustive des points forts et des points faibles de ces études, lesquels serviront comme repère lors de notre évaluation, suivi d'une courte présentation sur quelques projets incluant l'utilisation de bancs d'essai conçus spécifiquement pour l'évaluation de logiciels de virtualisation.

3.1 Présentation des travaux

Le tableau 3.1 présente un résumé des principales caractéristiques des études examinées : l'année de publication, le nom des logiciels de virtualisation évalués par chaque étude, la version de ces logiciels et du noyau de Linux utilisé (si mentionné dans l'étude) et les charges de travail des tests utilisées dans les expériences.

Étude	Année	Technologies comparées	Version / noyau	Charges de tests utilisées
[2]	2003	Xen VMware Workstation UML VMware ESX Server	– / 2.4.21 3.2 / 2.4.21 – / 2.4.21 – / 2.4.21	SPECcpu2000 INT, OSDB, Dbench, SPECweb99, compilation code source, LMbench, ttcp
[16]	2004	Xen	– / 2.4.26	Postmark, OSDB, SPECweb99, httperf, ttcp, dd compilation code source
[54]	2005	Linux-VServer UML Xen	1.29 / 2.4.27 – / 2.4.26, 2.6.7 – / 2.6.9	Mesure du temps de démarrage, de l'occupation mémoire, de l'espace disque utilisé, de la linéarité, de l'équité et de la sous-performance
[53]	2006	UML Linux-VServer Xen VMware Workstation VMware GSX	– / 2.4.26, 2.6.7 1.29 / 2.4.27 2.0 / 2.6.9 3.2 / 2.4.28 –	dd, Netperf, Mmesure du temps de démarrage, de l'occupation mémoire, de l'espace disque utilisé, de la linéarité, de l'équité et de la sous-performance
[20]	2006	Linux-VServer MCR[1] OpenVZ Xen	2.02rc9 / 2.6.15.4 2.1.0 / 2.6.14.4 2.5.1 / 2.6.15 022stab064 / 2.6.8 3.0.1 / 2.6.12.6	Dbench, LMbench, Tbench, compilation code source
[1]	2006	VMware Player[2]	1.0.1 / –	SPECcpu2000 INT, forkwait, SPECjbb2005, PassMark, compilation code source, ab, suite de *nano-benchmarks*
[36]	2006	VMware ESX Server	3.0 / –	LoadSim, SPECJbb2005, SPECweb2005, Swingbench, Dbench
[66]	2007	VMware ESX Server Xen	3.0.1 GA / – 3.0.3-0 / –	SPECcpu2000 INT, PassMark, Netperf, SPECjbb2005, compilation code source
[72]	2007	VMware ESX Server XenEnterprise	3.0.1 / 2.6.9 3.2[3] / 2.6.9	SPECcpu2000 INT, PassMark, Netperf, SPECjbb2005, compilation code source
[46]	2007	Xen OpenVZ	3.0.3 (*unstable*) / – *stable* / 2.6	RUBiS
[61]	2007	Linux-VServer Xen	2.0.1 / 2.6.17 3.0.2-testing / 2.6.16	LMbench, Iperf, OSDB, compilation code source, dd, Dbench, Postmark
[52]	2007	Linux-VServer	2.2 / 2.6.20	Mesure de la consommation de la mémoire et de la perte d'efficience en raison de la virtualisation
[41]	2007	Xen KVM	– / – 24 / –	LMbench, compilation code source

TAB. 3.1: Liste des études comparatives des technologies de virtualisation de serveurs publiées entre 2003 et 2007

[1] Projet expérimental utilisant une version modifiée du noyau de Linux avec une couche de « contention ».

[2] Avec et sans l'utilisation du support matériel à la virtualisation de l'architecture x86 (prototype).

[3] Xen Enterprise version 3.2 est basé sur la version 3.0.4 de Xen.

3.2 Charges de travail des tests utilisées

Avant de faire l'exposé des charges de travail des tests utilisées par les différentes études du tableau 3.1, il faut d'abord définir ce qu'on entend par l'expression charge de travail ainsi que par l'expression modèle de charge de travail employées par celles-ci :

> « La charge de travail (réelle) d'un système est composée de toutes les demandes de traitement, que ce soit un programme, des données ou des commandes, faites par les usagers. » Girard [18]

Dans ce contexte, une charge de travail des tests (de l'anglais *test workload*) peut être considérée comme un modèle de la charge réelle que subit une machine dans un environnement de production, soit l'ensemble des programmes qui sont conçus et implantés pour charger le système artificiellement pendant la prise de mesures [18].

Les charges de travail des tests utilisées par les études examinées peuvent être classées en deux catégories : les charges de tests synthétiques et les charges de tests artificielles.

Une charge de travail des tests synthétique est composée soit par un sous-ensemble des composants de base de la charge de travail réelle soit par un mélange de composants de la charge de travail réelle et de composants construits spécifiquement pour l'étude [18].

Contrairement aux charges de tests synthétiques, les charges de travail des tests artificielles ne contiennent aucun composant de la charge réelle [18]. Elles sont plutôt composées d'un ensemble de programmes conçus avec le seul but de surcharger une partie spécifique du système réel. Même si des charges de tests artificielles représentent un indicateur incomplet du comportement du système pour l'exécution d'une charge de travail réelle, elles offrent une opportunité d'observer l'impact des technologies de virtualisation sur l'exécution d'opérations primitives du système d'exploitation [61], comme la création de processus avec l'appel système *fork*.

La figure 3.1 présente un graphique de la distribution de l'occurrence des charges de travail des tests utilisées par les études examinées du tableau 3.1. Les sous-sections qui suivent font un bref exposé de chacune de ces charges de tests.

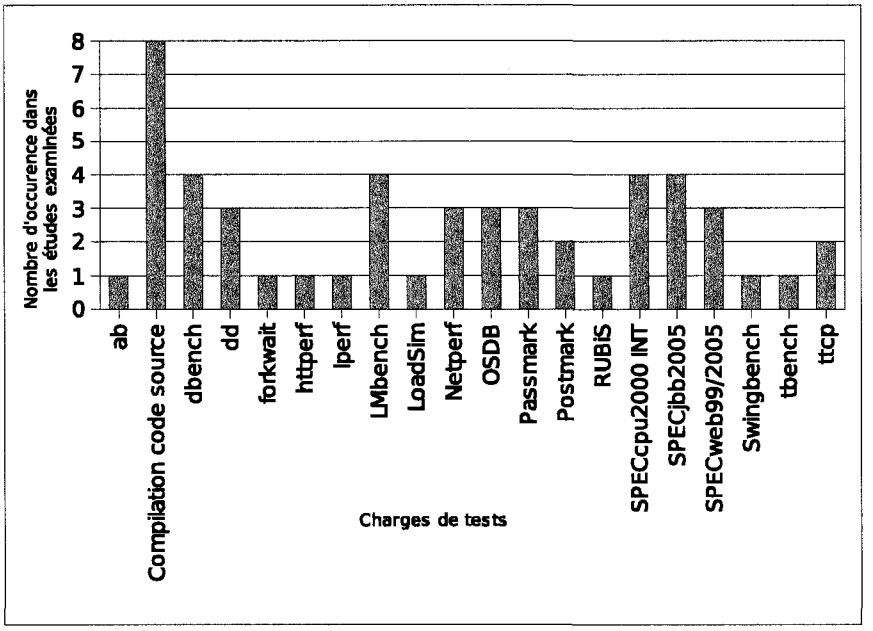

FIG. 3.1: Distribuition des occurrences des charges de tests utilisées par les études examinées

Les *bancs d'essai (benchmarks)*

Dans la littérature, les programmes conçus pour évaluer la performance d'un système sont appelés *benchmarks*, un mot anglais qui veut dire « point de référence » et qui peut être interprété comme « test de performance ». Ils sont souvent classés en trois catégories, en fonction de la nature de la charge de travail utilisée : les *macro-benchmarks*, les *micro-benchmarks* et les *nano-benchmarks*.

Les *macro-benchmarks* sont, en effet, des bancs d'essai. Ils sont utilisés dans l'évaluation de la performance au niveau des applications. Les *micro-benchmarks* se concentrent sur l'évaluation de la performance d'un composant du système en particulier, comme la mémoire ou le disque, tandis que les *nano-benchmarks* sont utilisés pour évaluer le système à un niveau encore plus bas. Ceux-ci ciblent l'exécution d'opérations sensibles à la virtualisation, comme les appels système à l'aide de l'instruction *syscall* [1].

3.2.1 Les charges de tests synthétiques

Cette section présente un bref exposé des charges de tests synthétiques utilisées par les études examinées.

Les bancs d'essai du groupe SPEC

SPEC est l'acronyme de *Standard Performance Evaluation Corporation*, le même pouvant être traduit comme « Société pour la normalisation de l'évaluation de la performance ». Il s'agit d'une organisation sans but lucratif régroupant plusieurs entreprises de l'industrie. Le but de cette organisation est d'établir des normes les plus objectives possibles pour mesurer la performance de plusieurs types de technologies [40]. Le groupe SPEC est particulièrement connu pour la création de bancs d'essai pour l'évaluation de serveurs, d'applications et de systèmes de haute performance.

Les bancs d'essai développés par la société sont identifiés par le préfixe *SPEC*, d'une référence à l'objet ciblé par l'évaluation et de son année de publication. La société ne supporte que la dernière version de ses bancs d'essai, lesquelles corrigent des problèmes d'impartialité (quand un logiciel est optimisé pour l'exécution du banc d'essai) identifiés par la communauté et qui sont adaptés à la puissance croissante des machines actuelles.

SPECcpu2000 INT La suite *SPECcpu* contient une série d'applications qui consomment énormément de cycles du processeur et ce pour une longue période de temps. Le suffixe INT fait référence au banc d'essai *CINT* qui fait partie de la suite et qui est composé de fonctions écrites dans le langage de programmation *C*. Ces fonctions exécutent une série des calculs avec des nombres entiers. La suite SPECcpu mesure la performance du processeur et de la mémoire du système.

Il s'agit d'une charge de travail « bénigne » [1], dans le sens que la majorité des instructions exécutées sont des instructions inoffensives (non sensibles).

SPECjbb2005 Le banc d'essai *SPECjbb* (*Java Server Benchmark*) évalue la performance d'un serveur d'applications Java. La charge de travail utilisée est composée d'une série d'applications exécutées par la JVM (la machine virtuelle Java). Il

s'agit d'un ensemble d'instructions qui contraint le processeur à travailler en mode d'opération non privilégié, de manière similaire à *SPECcpu INT* [1]. Le banc d'essai n'utilise pas le réseau et l'utilisation du disque se limite à l'écriture dans le journal d'événements du système [72] et à l'utilisation de la mémoire virtuelle.

SPECweb99/2000 *SPECweb* est le banc d'essai *SPEC* pour l'évaluation de la performance des serveurs web et des systèmes qui les accueillent. La charge de travail utilisée est composée d'un mélange complexe de requêtes de pages web avec des opérations HTTP et l'utilisation de contenu généré dynamiquement [16]. Le banc d'essai requiert l'utilisation de plusieurs machines clientes connectées au serveur de tests par réseau. Chacune de ces machines simulent un ensemble d'utilisateurs qui accèdent au serveur de manière concurrente [2].

Compilation de code source

La compilation du code source d'un programme inclut une quantité importante d'opérations d'entrée-sortie sur le disque, pour lire le code source et puis écrire le code compilé, une tâche qui exige beaucoup du système de fichiers [72]. De plus, c'est une tâche qui stresse l'UCT considérablement et, selon la structure du code source, peut entraîner l'utilisation de multiples fils d'exécution [61].

Un exemple de code source utilisé par les différentes études est le noyau de Linux. Les expériences mesurent le temps écoulé pour exécuter une compilation complète du code source du noyau sur le système d'exploitation hôte pour ensuite exécuter la même tâche sur le système d'exploitation invité. Ensuite, les résultats obtenus sont comparés. Ceci donne une idée du surcoût imposé par la couche de virtualisation.

dd

dd est un logiciel conçu pour faire la copie de données en bas niveau (bit par bit). Il a été utilisé par une partie des études examinées comme un outil de mesure pour analyser la capacité du système d'exécuter des opérations d'entrée-sortie sur le disque. L'expérimentation consistait à faire une copie d'un gros fichier dans le système de fichiers et à mesurer le temps nécessaire pour réaliser cette opération.

3.2.2 Les charges de tests artificielles

Cette section présente un bref exposé des charges de tests artificielles utilisées par les études examinées.

Dbench

Dbench est un banc d'essai utilisé pour évaluer la performance d'un serveur de fichiers. Il est dérivé du produit commercial *NetBench* et émule la charge imposée à un serveur de fichiers par des clients exécutant le système d'exploitation Windows 95 [2]. Le résultat affiché par le logiciel à la fin de son exécution représente le débit atteint par le serveur quand il est accédé par un seul client qui exécute environ 90000 opérations sur le système de fichiers [61].

LMbench

LMbench est une suite de bancs d'essai conçue pour mesurer les principaux goulots d'étranglement rencontrés dans une variété d'applications UNIX. Ceux-ci ont été identifiés par les développeurs de l'outil, puis isolés et reproduits par un ensemble de 37 *micro-benchmarks* [38]. Ils mesurent l'aptitude du système dans l'exécution de tâches concernant le transfert de données entre le processeur, la mémoire, le disque et le réseau, puisque la majorité des difficultés de performance vécues par un système est causée par des problèmes de latence et de bande passante, ou par une combinaison des deux [38].

Passmark

Passmark est une suite de bancs d'essai qui vise à évaluer, de manière isolée, plusieurs aspects de la performance du processeur d'un ordinateur utilisé comme poste de travail [1]. Les charges de tests ciblent l'ensemble des instructions non privilégiées de l'ISA du processeur.

Netperf

Netperf est un banc d'essai utilisé pour mesurer la performance de l'échange de données sur réseau. Il requiert l'utilisation d'un ou de plusieurs systèmes clients qui interagissent avec le serveur, tout en générant une charge sur le réseau. Le banc d'essai comprend autant de tests d'émission que de tests de réception de données.

ttcp

ttcp est la contraction de « test TCP », un outil pour mesurer la performance de la transmission et de la réception de paquets TCP par réseau. Il a été utilisé pour évaluer le surcoût imposé par la couche de virtualisation à l'utilisation du réseau [2].

Iperf

Iperf est un outil similaire à *ttcp* utilisé pour mesurer le débit maximal du lien réseau (la bande passante) en simulant un trafic de paquets TCP et UDP [61].

forkwait

forkwait est un programme simple conçu pour charger la procédure de création et de destruction de processus du système d'exploitation [1]. Il met l'emphase sur l'exécution des instructions sensibles utilisées par les appels système *fork* et *wait* des systèmes d'exploitation du type UNIX et aide à évaluer le surcoût imposé par la couche de virtualisation sur l'utilisation de l'UCT.

OSDB

OSDB, « *Open Source Database Benchmark* », est un banc d'essai libre conçu pour l'évaluation de la performance des serveurs de bases de données. Son mode de fonctionnement consiste à construire une base de données vide, à la remplir, pour ensuite exécuter une série d'opérations de recherche et de mise à jour des données, simulant ainsi la charge de plusieurs clients. La performance du système est mesurée en fonction de la quantité de transactions que le serveur est capable de gérer par seconde.

Tbench

Tbench est un banc d'essai complémentaire à *Dbench*. Pendant que Dbench se charge d'émuler la charge imposée au système de fichiers par le banc d'essai NetBench, tbench fait de même pour l'utilisation du réseau, en reproduisant le trafic des paquets TCP créés lors de l'exécution de *NetBench* [63].

ab

ab est l'acronyme pour « *Apache Benchmark* », un outil conçu pour évaluer la performance d'un serveur web Apache et qui est inclu dans le paquet d'installation de ce dernier. Il est utilisé pour déterminer le nombre de requêtes HTTP qu'une installation de ce serveur est capable de gérer par seconde.

RUBiS

RUBiS est l'acronyme de *« Rice University Bidding System »*, un banc d'essai conçu pour évaluer la performance et la capacité d'expansion des sites web de vente aux enchères. L'application est composée d'un serveur web, qui accueille le site web, et d'un serveur de bases de données, qui enregistre les produits et qui garde une trace des opérations effectuées. Il est possible de configurer plusieurs clients *RUBiS* qui se connectent au serveur d'applications et qui exécutent des opérations simulant les actions des utilisateurs d'un tel système, comme la recherche, l'achat et la vente de produits [46].

httperf

httperf est un outil conçu pour mesurer la performance d'un serveur web. Il est possible de le configurer de manière à lui faire générer une charge de travail fixe ou variable constituée des requêtes HTTP qui sont envoyées au serveur cible. La mesure des résultats est faite sur le temps nécessaire au serveur pour répondre aux requêtes.

SwingBench

SwingBench est une suite logicielle conçue pour évaluer la performance des bases de données Oracle. Elle est composée, entre autres, d'un générateur de charge et de quatre bancs d'essai. La métrique utilisée pour évaluer la performance du système est le nombre de transactions effectuées sur la base de données par seconde [36].

LoadSim

LoadSim (contraction de « Load Simulator ») est un générateur de charge accompagné d'un outil de mesure conçu pour évaluer la performance des serveurs de courriel Microsoft Exchange [36].

Postmark

Postmark est un banc d'essai conçu pour émuler la charge de travail d'un serveur de courriel ou de nouvelles très chargé [61]. Pour réaliser cette émulation, il charge le système de fichiers en exécutant plusieurs opérations sur des fichiers de petite taille [16].

3.3 Analyse des points forts et des points faibles des études examinées

Indépendamment de son objectif central, toutes les études mentionnées dans le tableau 3.1 présentent des résultats d'expériences exécutées dans le but d'évaluer ou de comparer différents logiciels de virtualisation. Certaines de ces études [53, 61, 46] sont très complètes à ce niveau et présentent en détails la méthodologie utilisée ainsi qu'une discussion étendue sur les résultats obtenus. D'autres études [41, 20] se limitent à la présentation des résultats accompagnée d'une brève explication de la procédure utilisée pour la prise de mesures. Cette section met en évidence certains aspects importants à mentionner au sujet des quelques caractéristiques des études examinées.

3.3.1 Choix des métriques et des charges de tests

La majorité des études ont utilisé des charges de tests communément employées pour l'évaluation de la performance d'un système, comme les bancs d'essai du groupe SPEC, Dbench et LMbench. Dans cette section, nous commentons trois études qui ont choisi des approches différentes.

Quétier *et al.* [54, 53] évaluent les technologies de virtualisation en fonction de leur « utilisabilité » (de l'anglais *usability* qui s'intéresse à la capacité des machines virtuelles de co-habiter dans une même machine physique) en plus de leur performance. Au niveau de la utilisabilité, les auteurs ont exploré certains facteurs qui limitent le nombre de machines virtuelles pouvant être lancées simultanément [54], comme le temps de démarrage, l'occupation mémoire et l'espace disque utilisé. Les métriques choisies pour l'évaluation de la performance sont la linéarité, l'équité et la sous-performance des technologies de virtualisation. Les auteurs explorent aussi la communication entre les machines virtuelles ainsi que la *granularité* [54] des technologies de virtualisation (ou leur capacité de mise à l'échelle) et ce en fonction des changements de contexte (le temps requis par le système hôte pour ordonnancer une autre machine virtuelle) et de la tranche de temps de l'UCT accordée à chaque machine virtuelle.

Padala *et al.* [46] évaluent de manière quantitative l'impact des technologies de virtualisation dans le processus de consolidation des serveurs au niveau des applications. L'étude utilise RUBiS (voir Section 3.2.2) comme banc d'essai et évalue comment la performance de l'application (mesurée en fonction du débit atteint ainsi que du temps de réponse) est affectée par l'utilisation de différentes technologies de virtualisation et ce avec différentes configurations de consolidation (nombre de machines virtuelles par serveur physique). Ils ont découvert, par exemple, que la mise à l'échelle de l'application (l'exécution de plusieurs instances de RUBiS), est simplifiée (en termes de temps de réponse et d'utilisation de l'UCT) si le serveur web et le serveur de base de données qui composent chaque instance de RUBiS sont installés dans des machines virtuelles distinctes, exécutées sur différents serveurs physiques, au lieu d'être installés dans une même machine virtuelle.

3.3.2 Validité des résultats

Un des principaux problèmes rencontrés dans la majorité des études comparatives est le manque d'impartialité des auteurs. Dans huit [2, 16, 1, 36, 66, 72, 61, 52] des treize études examinées au moins un des auteurs est impliqué dans le développement de l'une des technologies de virtualisation ciblées par l'étude ou travaille pour la compagnie qui détient les droits sur la technologie. Cela ne veut pas forcement indiquer que les résultats de ces études sont douteux, mais il y a des cas où le conflit d'intérêt est évident.

VMware [66] compare un de ses logiciels de virtualisation de serveurs avec un autre logiciel d'une compagnie concurrente. Après la réalisation de plusieurs expériences ils concluent que leur produit est supérieur à celui de la concurrence dans tous les cas d'utilisation. Les résultats ont été contestés par XenSource [72] qui a reproduit les expériences en arrivant à des résultats différents de ceux de l'étude originale.

Cet exemple démontre l'importance de bien décrire la méthodologie utilisée dans l'étude ainsi que tous les détails qui peuvent aider à la reproduction des expériences.

3.3.3 Autres considérations

Cette section introduit d'autres points intéressants sur quelques-uns des travaux présentés.

La conception de Xen a beaucoup changé depuis que la technologie a été évaluée par Barham [2] en 2003. Un exposé général de la génération courante est présenté par Soltesz *et al.* [61]. Cependant, l'article original reste une des principales références dans le domaine des études comparatives et est cité par dix des douze autres articles révisés.

Soltesz *et al.* [61] et Pötzl & Fiuczynski [52] ont mésurée l'efficience des technologies de virtualisation en fonction de leur performance ainsi que de leur capacité de mise à l'échelle. Il s'agit d'un choix intéressant. Les auteurs affirment que la combinaison de ces deux métriques correspond directement à la manière dont le VMM gère les ressources disponibles dans le système pour répondre aux besoins des machines virtuelles et que, pour cette raison, elle constitue un bon indicatif de son efficience.

La majorité des études examinées évaluent une quantité limité des logiciels de

virtualisation, généralement restreint à un [16, 1, 36, 52] ou deux [66, 72, 46, 61, 41] logiciels. Aucune étude ne présente une comparaison de l'ensemble des principales technologies de virtualisation utilisées.

3.4 Bancs d'essai spécifiques à la virtualisation

La virtualisation des serveurs x86 est très récente et l'industrie cherche encore la meilleure méthodologie à utiliser dans l'évaluation des différentes technologies de virtualisation. Cette section présente trois approches : la création d'un comité « virtualisation » organisé par l'organisme SPEC et les projets *vConsolidate* de Intel et *VMmark* de VMware.

3.4.1 Le Comité de virtualisation SPEC

En novembre de 2006, le groupe SPEC a créé un groupe de travail afin d'évaluer la pertinence d'un nouveau banc d'essai conçu spécifiquement pour évaluer la performance des technologies de virtualisation [40]. Ce groupe a été baptisé « Comité de virtualisation SPEC » et est composé de plusieurs représentants de l'industrie tels que AMD, Intel, HP, IBM, Sun Microsystems, Dell, Microsoft, Red Hat, SWsoft et VMware, entre autres.

Ce comité a décidé de créer un nouveau banc d'essai et analyse actuellement la charge moyenne des principales applications rencontrées dans les centres de données [10]. Cette évaluation doit leur permettre d'identifier les paramètres nécessaires au développement du nouveau banc d'essai, prévu pour la deuxième moitié de 2008 [10]. L'idée de base de ce projet est de développer un outil « standard » pour l'industrie, ce qui permettra aux utilisateurs d'évaluer eux-mêmes la performance des logiciels de virtualisation déployés avec un nombre varié de machines virtuelles exécutant différentes charges de travail [62].

3.4.2 *vConsolidate*

Intel, plutôt que d'attendre les résultats du Comité de virtualisation SPEC, a développé un « guide de bonnes pratiques » accompagné d'une méthodologie générale

pour le développement d'un banc d'essai pour la virtualisation appelée *vConsolidate* [9]. L'objectif de la compagnie avec vConsolidate est de proposer une méthodologie pour comparer différentes plateformes et technologies de virtualisation. Le tout est basé sur la proposition suivante :

> « Un banc d'essai pour la virtualisation doit évaluer la performance générale d'un système composé de plusieurs charges de travail différentes consolidées sur un même serveur physique, mais exécutées sur différentes machines virtuelles. » [9]

Cette proposition est en accord avec Makhija [36] qui affirme que les bancs d'essai traditionnels, comme ceux développés par SPEC, n'ont pas été conçus pour l'évaluation des machines virtuelles ou d'un système qui fait la consolidation de charges de travail. Ceux-ci mettent l'emphase sur l'exécution d'une seule charge de travail et ont été développés pour surcharger le serveur jusqu'au point de saturation. Si la majorité des charges de travail réelles avaient ce comportement, la consolidation des serveurs ne serait pas nécessaire, car l'exécution d'une seule charge de travail serait suffisante pour utiliser toutes les ressources du serveur.

Intel propose d'utiliser une charge de travail des tests composée de plusieurs charges de travail différentes de manière à simuler, par exemple, un système qui accueille en même temps deux serveurs de bases de données, quatre serveurs web et un serveur de courriel. La performance de chaque type de serveur est évaluée par un banc d'essai spécifique à la charge de travail du serveur cible [4], comme OSDB pour les serveurs de base de données, SPECweb2005 pour les serveurs web et LoadSim pour le serveur de courriel.

La première étape consiste à exécuter chacun de ces bancs d'essai individuellement dans un environnement non virtualisé standard. Les résultats obtenus constituent les lignes de base pour ces bancs d'essai. Ensuite, ils doivent être exécutés de nouveau, mais cette fois dans un environnement virtualisé. Les résultats obtenus doivent être normalisés par les lignes de base respectives :

résultat normalisé = résultat environnement virtualisé / ligne de base

[4] Il n'est pas clair dans l'article de Rosenblum [9] quels sont les bancs d'essai qui composent la suite *vConsolidate* ni si elle est disponible pour l'évaluation par le public.

Ainsi, si le résultat de l'exécution de OSDB dans un environnement virtualisé donne 1800 transactions/seconde, celui de LoadSim donne 1500 courriels/seconde et celui de SPECweb2005 donne 2300 transactions/seconde, et si les lignes de base pour ces bancs d'essai sont respectivement 1950, 1700 et 2450, les résultats normalisés seront 0,923 pour le serveur de base de données, 0,882 pour le serveur web et 0,939 pour le serveur de courriel.

Pour obtenir le résultat final de vConsolidate pour la configuration de consolidation mentionnée il faut multiplier chacun des résultats normalisés obtenus par un « poids » qui correspond au nombre de serveurs du même type divisé par le nombre total de serveurs, pour ensuite additionner les résultats de ces calculs. Pour notre exemple, le résultat final, ou « score » vConsolidate, est :

$$\textit{résultat final} = (0,923 * (2/7)) + (0,882 * (4/7)) + (0,939 * (1/7)) = 0,902$$

3.4.3 *VMmark*

Au moment de la création du Comité de virtualisation SPEC, un autre de ses membres, VMware, avait déjà un outil pour l'évaluation de logiciels de virtualisation, appelé *VMmark*. Cet outil était, à ce moment, un banc d'essai approuvé par la société SPEC [62].

L'objectif de VMmark est d'évaluer la performance générale d'une infrastructure informatique virtualisée. Il cherche aussi à vérifier combien de machines virtuelles le logiciel de virtualisation utilisé est capable d'exécuter en même temps et dans quelle proportion ce nombre augmente à mesure qu'on augmente les ressources physiques du serveur (principalement la mémoire) [36]. Cet outil partage les mêmes principes de base que *vConsolidate* ; les différences sont au niveau de la charge de tests, de l'unité de mesure et de la méthodologie choisie.

La charge de tests est composée de cinq machines virtuelles qui exécutent chacune une charge de travail spécifique plus une machine virtuelle qui n'exécute aucune charge de travail. Cette dernière a été incluse pour représenter un serveur « en attente », un scénario fréquemment rencontré dans les centres de données selon VMware. Même si ce serveur est utilisé seulement en cas de besoin, il consomme tout de même une petite

Serveur virtuel	Banc d'essai
Serveur de courriel	*LoadSim*
Serveur Java	SPECjbb2005
Serveur Web	SPECweb2005
Serveur de bases de données	*Swingbench*
Serveur de fichiers	*Dbench*
Serveur en attente	–

TAB. 3.2: Serveurs virtuels et bancs d'essai utilisés par VMmark comme charge de tests.

partie des ressources disponibles. La performance des cinq autres serveurs virtuels est évaluée avec un banc d'essai spécifique pour ces serveurs, de manière similaire à vConsolidate. La relation entre les serveurs et les bancs d'essai respectifs utilisés par VMmark est présentée dans le tableau 3.2.

L'ensemble de ces six serveurs virtuels compose une sorte « d'unité de charge de travail de base », ce que Makhija [36] appelle « tuile » (de l'anglais *tile*). Des *tuiles* sont ajoutées progressivement à la charge de test tant qu'il existe des ressources physiques disponibles. Le nombre total de *tuiles* qu'un serveur physique doté de la couche de virtualisation est capable d'accueillir donne la mesure de la capacité de consolidation du système. Le résultat général de VMmark est déterminé par une fonction qui utilise comme paramètres le nombre de *tuiles* supporté par le système et les résultats individuels des bancs d'essai utilisés pour évaluer chaque serveur virtuel ainsi que par un système de normalisation semblable à celui adopté par vConsolidate. Les détails de ce calcul sont présentés dans le guide d'utilisation du logiciel [67].

La principale limitation de cet outil est qu'il n'a pas été conçu pour évaluer la performance des logiciels de virtualisation en général, comme on pourrait s'attendre d'un banc d'essai qui envisage de devenir « la norme de l'industrie ». En effet, il a été conçu pour évaluer le principal produit de VMware : le logiciel de virtualisation VMware ESX Server. En pratique, VMmark est utilisé par les consommateurs de VMware pour comparer la performance de VMware ESX Server sur différents serveurs physiques et pour déterminer lequel est le plus approprié pour accueillir leur charge de travail.

Chapitre 4

Méthodologie

Comme présenté dans le chapitre précédent, il n'existe pas de consensus dans la communauté scientifique ni l'industrie sur la meilleure méthode à employer pour l'évaluation des logiciels de virtualisation. La majorité des études présentées dans le tableau 3.1 ont utilisé une combinaison variée de bancs d'essai qui ciblent différentes parties du système virtualisé. Nous avons utilisé dans cette étude une approche similaire.

Ce chapitre décrit la méthodologie employée lors de l'exécution des expériences comparatives. Il se divise en deux parties. La première partie décrit la procédure utilisée pour l'évaluation des logiciels de virtualisation. La deuxième partie décrit le matériel utilisé pour l'exécution des expériences, les configurations spécifiques à chaque technologie de virtualisation ainsi que les différents paramètres utilisés pour chaque expérience.

4.1 Procédure d'évaluation

L'efficience des logiciels de virtualisation est évaluée en fonction de leur performance ainsi que de leur capacité de mise à l'échelle. Ce modèle d'analyse est suggéré par Soltesz *et al.* [61] car la combinaison de ces deux métriques constitue un bon indicatif de la manière dont la couche de virtualisation gère les ressources physiques de la machine pour répondre aux besoins des machines virtuelles.

Notre analyse a été faite en deux étapes. La première étape a pour but d'évaluer

Banc d'essai	Version (hôte)	Version (VMs)	Unité de mesure
Bzip2	1.0.3-0ubuntu2.1	1.0.4-0ubuntu2.1	temps
Dbench	3.04	3.04	débit
Dd (coreutils)	5.93-5ubuntu4	5.97-5.3ubuntu3	débit
Compilation noyau	linux-2.6.22.14	linux-2.6.22.14	temps
Netperf	2.4.4	2.4.4	débit
Rsync	2.6.6-1ubuntu2.1	2.6.9-5ubuntu1	temps
Sysbench	0.4.8	0.4.8	débit

TAB. 4.1: Bancs d'essai utilisés pour l'évaluation de performance et unités de mesure respectives

la performance des logiciels de virtualisation. Pour ce faire, nous avons utilisé une série de bancs d'essai qui ont été exécutés sur un environnement Linux non virtualisé (Linux *natif*) puis sur l'environnement virtuel offert par les logiciels de virtualisation. La différence entre les résultats nous donne une indication du surcoût imposé par les couches de virtualisation. La deuxième étape a évalué la capacité de mise à l'échelle des logiciels de virtualisation, soit l'habilité du système d'exploitation hôte à répondre efficacement aux demandes de plusieurs machines virtuelles exécutant la même charge de travail de manière concurrente.

4.1.1 Évaluation de la perte d'efficience

Nous avons utilisé une combinaison de sept expériences différentes pour déterminer le surcoût imposé par la couche de virtualisation : l'exécution des bancs d'essai Netperf, Dbench et Sysbench, la compilation du noyau de Linux, la copie de données avec le logiciel dd, l'utilisation du logiciel Bzip2 pour la compression de fichiers et le transfert de données par réseau avec le logiciel Rsync. Les versions des bancs d'essai et des utilitaires du système d'exploitation utilisés dans les expériences sont présentées par le tableau 4.1. Comme nous avons utilisé, dans les machines virtuelles, une version du système d'exploitation différente de celle de l'hôte (voir explication dans la section 4.2.3), il n'a pas toujours été possible d'utiliser la même version des bancs d'essai dans toutes les expériences.

Dans un premier temps, nous avons exécuté les expériences sur le système d'exploitation hôte configuré sans aucune couche de virtualisation. Les résultats obtenus sont utilisés comme base de comparaison pour les calculs de normalisation. Nous appelons ce résultat le résultat de base ($Rbase$). Ensuite, les mêmes expériences sont exécutées sur un système d'exploitation invité. À ce moment, seulement une machine virtuelle s'exécute sur l'hôte - celle qui a accueilli le système d'exploitation invité.

Chaque expérience est exécutée quatre fois. Les résultats de la première série (RS_0) sont ignorés ; ils servent à « échauffer » le système (remplir les caches). La moyenne des résultats de la deuxième, de la troisième, ainsi que de la quatrième série est utilisée comme résultat représentatif (rr), comme l'indique l'équation suivante :

$rr = (RS_1 + RS_2 + RS_3)/3$

Ensuite, les résultats représentatifs de chaque expérience sont normalisés par rapport aux résultats de base respectifs de chaque expérience :

$r = rr/Rbase$

La différence entre le résultat représentatif et le résultat de base respectif indique la sous-performance de la technologie de virtualisation : plus elle est proche de un, plus faible est la perte d'efficience. Un valeur plus grande que un indique que l'expérience a présenté une meilleure performance lorsqu'elle est exécutée sur l'environnement virtuel.

4.1.2 Évaluation de la capacité de mise à l'échelle

L'utilisation de la virtualisation pour la consolidation de serveurs implique l'exécution de plusieurs machines virtuelles en même temps. Pour examiner la capacité de mise à l'échelle des logiciels de virtualisation, nous avons utilisé le banc d'essai Sysbench. Il a été exécuté de manière concurrente sur 1, 2, 4, 8, 16 et puis 32 machines virtuelles.

Habituellement, une machine utilisée dans le processus de consolidation de serveurs accueille différentes applications qui exécutent des charges de travail distinctes. Nous avons choisi d'exécuter la même charge de travail sur l'ensemble des machines virtuelles avec pour objectif de vérifier si les ressources du serveur physique qui les accueille sont partagées équitablement entre elles.

Cette expérience a été exécutée six fois pour chaque logiciel de virtualisation, en commençant avec l'exécution d'une seule machine virtuelle puis en doublant le nombre de machines virtuelles à chaque nouvelle expérience. De manière similaire à l'évaluation de la performance, chaque expérience est composée de quatre séries d'exécutions, le résultat de la première série n'étant pas considéré dans le calcul du résultat représentatif.

4.2 Matériel et paramètres de configuration

Cette section décrit le matériel utilisé pour les expériences, la configuration des machines virtuelles ainsi que les versions des logiciels utilisés pour cette étude. L'annexe A présente la liste complète des commandes et des paramètres utilisés dans l'exécution des bancs d'essai ainsi que l'information détaillée sur l'installation des logiciels de virtualisation, la création et le démarrage des machines virtuelles.

4.2.1 Configuration des machines physiques

Pour les besoins de nos expériences, nous avons utilisé comme serveur de tests un ordinateur de bureau IBM/LeNovo configuré avec un processeur Intel Core 2 Duo 6300, 4 Go de mémoire vive, un disque dur SATA de 80 Go et deux interfaces réseau de un gigabit. L'interface réseau primaire est branchée sur le réseau local alors que l'interface réseau secondaire est branchée à un deuxième ordinateur, utilisé comme machine cliente pour les expériences qui impliquent l'échange de données sur le réseau. Ce deuxième ordinateur est un Dell Optiflex GX configuré avec un processeur Intel Pentium 2, 123 Mo de mémoire vive, un disque dur de 250 Go et une interface réseau de un gigabit.

Lors des expériences sur la perte d'efficience, un troisième ordinateur, branché au réseau local, est utilisé pour accéder au serveur de tests et démarrer l'exécution des bancs d'essai. Il s'agit d'un ordinateur portable IBM Thinkpad configuré avec une interface réseau de un gigabit. Pour les expériences de mise à l'échelle, ce même ordinateur a été branché au réseau privé. Il a été utilisé pour établir des connexions SSH avec les machines virtuelles et démarrer simultanément l'exécution des bancs

Numéro de MV (n)	Quantité de mémoire par MV (en Mo)
n=1	2039
n=2	1622
n=4	811
n=8	405
n=16	202
n=32	101

TAB. 4.2: Allocation de mémoire par machine virtuelle utilisée lors de l'évaluation de mise à l'échelle

d'essai (voir section 4.3.1).

4.2.2 Configuration des machines virtuelles

Lors de l'évaluation sur la perte d'efficience, les machines virtuelles sont configurées avec 2 Go de mémoire vive. Pour l'évaluation de mise à l'échelle, nous réservons au système d'exploitation hôte 512 Mo de mémoire vive et nous partageons la mémoire restante de manière équitable entre les n machines virtuelles en exécution, comme le montre le tableau 4.2. Nous avons fixé à 2039 Mo la quantité maximale de mémoire vive pouvant être allouée à une machine virtuelle car il n'a pas été possible d'exécuter les machines virtuelles basées sur QEMU avec plus de mémoire. Cette quantité de mémoire est, portant, plus que suffisante pour l'exécution des bancs d'essai utilisés par cette étude ainsi que pour l'exécution de la grosse majoritée des logiciels utilisés couramment dans le marché.

4.2.3 Choix du système d'exploitation

La distribution Linux Ubuntu est le système d'exploitation choisi pour les expériences. Habituellement, les serveurs en production utilisent la version LTS (de l'anglais *Long Time Supported*) d'Ubuntu qui à l'époque de cette évaluation était la version numéro 6.06. Cette version spéciale est maintenue par le distributeur avec des mises à jour de sécurité pour une période de cinq ans. Pour cette raison, nous l'avons

Logiciel	Version	Noyau S.E. hôte	Noyau S.E. invité
KQEMU	1.3.0~pre11-6	2.6.22.14-kqemu	2.6.15-26-amd64
KVM	58	2.6.22-14-server	2.6.15-26-amd64
Linux-VServer	2.2.0.5	2.6.22.14	2.6.22.14
OpenVZ	5.1	2.6.22-ovz005	2.6.22-ovz005
VirtualBox	1.5.4_OSE / 1.5.51_OSE	2.6.22-14-server	2.6.22-14
Xen	3.1.0	2.6.22-14-xen	2.6.22-14-xen

TAB. 4.3: Logiciels de virtualisation évalués ainsi que les versions respectives du noyau de Linux utilisées dans les expériences

choisie comme système d'exploitation pour les machines virtuelles.

Malheureusement, le noyau de Linux inclus dans cette version d'Ubuntu n'est pas supporté par tous les logiciels de virtualisation qui font l'objet de notre évaluation. Pour cette raison, nous avons choisi une version plus récente d'Ubuntu, la version 7.10, comme système d'exploitation pour la machine de tests. La machine cliente a été configurée avec la version LTS d'Ubuntu comme système d'exploitation.

4.2.4 Configuration des logiciels de virtualisation

Le tableau 4.3 présente la version des logiciels de virtualisation et les versions respectives du noyau de Linux utilisées dans le système d'exploitation hôte et dans le système d'exploitation invité pour chaque expérience. L'annexe A contient des informations détaillées sur l'installation et l'utilisation des logiciels de virtualisation.

Il n'a pas été possible d'installer la version x86_64 (amd64) d'Ubuntu dans les machines virtuelles VirtualBox car elles ne supportent pas le « long mode » (64 bits). Ces machines virtuelles fonctionnent en mode 32 bits indépendemment de l'architecture de la machine hôte. Pour cette raison, les tests avec VirtualBox ont été exécutés en mode 32 bits tandis que les autres tests ont été faits en mode 64 bits. Comme l'utilisation de mémoire par machine virtuelle ne dépasse pas la barrière des 4 Go, l'exécution des machines virtuelles en mode 32 bits ne répresente pas un handicap majeur pour VirtualBox.

4.3 Prise de mesures

Cette section explique la procédure utilisée pour accéder au serveur de tests et démarrer l'exécution des bancs d'essai. Elle présente aussi les métriques utilisées dans les expériences.

4.3.1 Accès au serveur et exécution des bancs d'essai

Pour les expériences sur la perte d'efficience, l'accès au serveur de tests (le serveur hôte) a été fait à travers une connexion distante utilisant le logiciel SSH. Dans cette étape, seulement l'hôte a été configuré avec un serveur SSH. Les machines virtuelles sont accédées au moyen de la connexion établie avec l'hôte et les différents outils de gestion qui accompagnent les logiciels de virtualisation. Il n'y a aucune connexion locale sur le serveur hôte pendant l'exécution des expériences. Seulement une connexion SSH a été établie en tout temps sauf pour les tests impliquant le réseau (Netperf et Rsync), où une seconde connexion est établie entre le serveur de tests et la machine cliente (celle-ci n'étant accessible que par le réseau privé).

Pour les expériences de mise à l'échelle, les machines virtuelles ont été configurées avec un serveur SSH. Le logiciel d'émulation de terminaux *Konsole* a été utilisé pour établir les connexions SSH (sessions) directement avec les machines virtuelles. L'exécution simultanée du banc d'essai *Sysbench* sur les n machines virtuelles a été déclenchée à l'aide de l'option « Envoyer la saisie à toutes les sessions » du logiciel Konsole. La procédure utilisée, illustrée par la figure 4.1, est la suivante :

1. Exécuter le logiciel *Konsole* sur la machine cliente (l'ordinateur portable).
2. Ouvrir n sessions (ou n est égal au nombre de machines virtuelles ciblées par l'expérience).
3. Sur le menu de la première session, sélectionner l'option « Envoyer la saisie à toutes les sessions ». Cela indique que les prochaines commandes exécutées dans cette session seront aussi exécutées dans les autres sessions ouvertes.
4. Obtenir le numéro d'identification de la session et l'exporter dans une variable appelée « n » : `export n=‘echo $KONSOLE_DCOP_SESSION |awk -F - ’{ print $3 }’ |cut -d’)’ -f1‘`

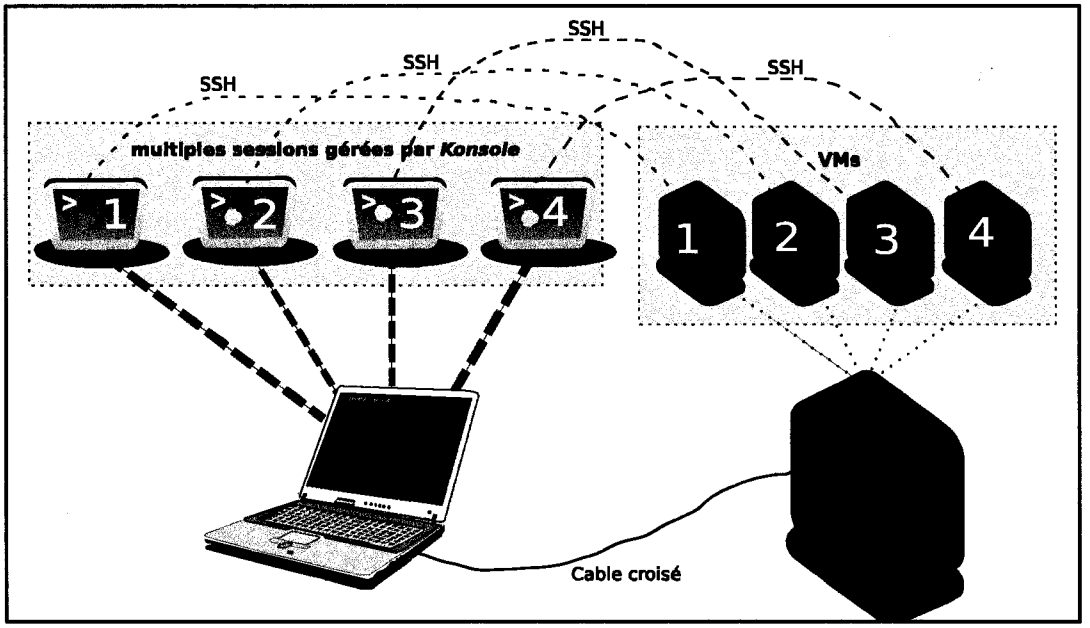

FIG. 4.1: Exécution de l'expérience de mise à l'échelle avec quatre machines virtuelles à l'aide du logiciel Konsole

5. Ouvrir une connexion avec chaque machine virtuelle. Le nom des machines virtuelles est toujours composé par le prefix « vm » plus un numéro d'identification (1 à 32). La commande suivante fait que chacune des sessions Konsole ouvre une connexion SSH avec une machine virtuelle différente : `ssh root@vm$n`
6. Exécuter le banc d'essai *Sysbench* (la commande exacte est fournie dans l'annexe A).

À partir de l'étape 4 de la procédure précédente, les commandes sont exécutées simultanément sur toutes les sessions. Les machines virtuelles utilisées sont identiques ; il s'agit d'une réplique exacte de la machine virtuelle utilisée pour les tests de perte d'efficience, déjà configurée pour le banc d'essai Sysbench, les seules altérations étant le nom du serveur (*hostname*) et la présence d'un serveur SSH. Les adresses IP des machines virtuelles sont connues par le DNS.

4.3.2 Unités de mesure

Deux unités de mesures sont utilisées lors de nos expériences : le temps d'exécution et le débit.

Pour trois des tests, compilation du noyau, Rsync et Bzip2, la métrique utilisée est le temps d'exécution. Nous avons utilisé la commande « *date* » du système d'exploitation Linux pour mesurer le temps d'exécution. La commande est utilisée avec l'argument « *+%s.%N* » qui donne le temps en secondes, avec une précision en nano-secondes, en fonction d'un moment fixe dans le passé utilisé comme référence. Cette commande est employée deux fois : une première fois immédiatement avant l'exécution de l'expérience et une seconde fois immédiatement après l'exécution de l'expérience. Les deux commandes sont jumelées en une seule de la façon suivante :

```
date +%s.%N && <commande_pour_l'exécution_du_banc_d'essai> && date
+%s.%N
```

La différence entre les valeurs résultantes de la première et de la seconde exécution de la commande *date* est utilisée pour déterminer le temps d'exécution (la durée) de l'expérience.

Pour les autres tests, Dbench, Netperf, dd et Sysbench, la métrique utilisée est le débit. L'utilisation de cette métrique est simple car nous avons utilisé comme résultat les valeurs affichées par la sortie d'exécution des bancs d'essai eux-mêmes.

Le tableau 4.1 montre les métriques utilisées pour chaque expérience.

Chapitre 5

Résultats et discussion

Ce chapitre présente les résultats de notre évaluation sur l'efficience des six logiciels de virtualisation suivants : KQEMU, KVM, Linux-*VServer*, OpenVZ, Xen et VirtualBox. Comme indiqué dans le chapitre précédant, nous avons évalué la performance des logiciels de virtualisation en fonction de la perte d'efficience qu'ils engendrent ainsi que de leur capacité de mise à l'échelle. En conséquence, ce chapitre est divisé en deux sections. Nous avons opté de commenter les résultats au fur et à mesure de leur présentation.

5.1 Évaluation de la perte d'efficience

Cette section présente les résultats de la première étape de notre évaluation, la perte d'efficience engendrée par les logiciels de virtualisation. Les résultats des différentes expériences sont normalisés par rapport aux résultats obtenus avec Linux natif. Dans les graphiques suivants, les résultats obtenus par Linux natif sont représentés par le chiffre « 1 ». La sous-performance des logiciels de virtualisation est exprimée en fonction de cette référence : plus elle est proche de « 1 », moins importante elle est. Une valeur supérieure à « 1 » indique que l'expérience a présentée une meilleure performance lorsqu'elle est exécutée sur l'environnement virtuel.

Le logiciel de virtualisation VirtualBox permet aux utilisateurs de choisir d'utiliser ou non les extensions additionnelles incorporées à l'architecture x86 pour « faciliter » la virtualisation (voir section 1.3.1). Cette option nous a conduit à exécuter toutes

les expériences avec VirtualBox deux fois, avec (*-hwvirtex on*) et sans (*-hwvirtex off*) l'emploi de ces extensions.

Comme précisé dans le chapitre 2, le logiciel d'émulation QEMU est utilisé comme base pour un nombre considérable de logiciels de virtualisation évalués dans cette étude. Tant KVM que KQEMU utilisent l'image de la machine QEMU comme source pour ses machines virtuelles, la différence étant la manière dont ces deux logiciels de virtualisation gèrent l'exécution des instructions. Pour cette raison, même s'il ne s'agit pas d'un logiciel de virtualisation, nous avons décidé d'évaluer aussi la performance de QEMU dans la première partie de notre étude, l'objectif étant de vérifier comment une couche de virtualisation appliquée au-dessous de QEMU (comme celle utilisée par KVM et KQEMU) peut améliorer la performance des applications exécutées par ce dernier.

5.1.1 Compilation du noyau de Linux

La figure 5.1 présente les résultats pour l'expérience de compilation du noyau de Linux. La charge de travail est centrée sur une utilisation intensive de l'UCT et implique l'exécution de plusieurs fils, en plus de charger le système de fichiers avec la lecture et l'écriture de petits fichiers. Ces caractéristiques font de cette expérience un bon indicateur de la performance générale du système.

Comme prévu, les logiciels qui emploient la paravirtualisation ou qui font de la virtualisation au niveau du système d'exploitation présentent une performance proche de celle de Linux natif. Parmi les logiciels employant la virtualisation complète, la performance de KVM s'est montrée supérieure aux autres. Comparé à Linux natif, QEMU prend vingt fois plus de temps pour compléter la tâche de compilation du noyau.

Le graphique de la figure 5.1 fait ressortir une situation unique dans notre étude. En effet, la non-utilisation des extensions additionnelles à l'architecture x86 par VirtualBox produit une performance significativement supérieure à celle obtenue lorsque ce même logiciel utilise ces extensions pour exécuter la même charge de travail. Dans toutes les autres expériences, présentées dans les sections qui suivent, cette différence de performance est moins significative.

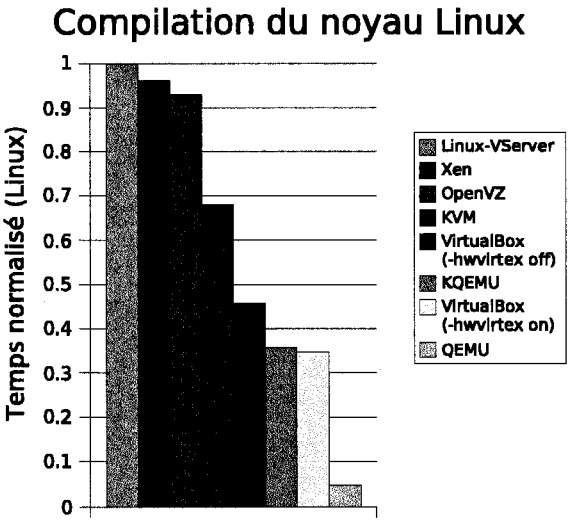

FIG. 5.1: Performance des logiciels de virtualisation pour la compilation du noyau de Linux

5.1.2 Compression de fichier avec Bzip2

L'expérience suivante est la compression d'un fichier contenant une image *iso* de 433 Mo avec le logiciel Bzip2. La compression de fichiers est, comme la compilation de code, une tâche centrée dans l'utilisation intensive de l'UCT, mais avec un faible taux d'opérations d'entrée-sortie. L'option « -9 », utilisée pour obtenir une compression maximale (voir l'annexe A pour la commande complète), demande aussi plus de mémoire pour l'exécution du processus.

Pour cette expérience, presque tous les logiciels de virtualisation ont présenté une performance proche de celle de Linux natif, à l'exception de KQEMU et de OpenVZ, comme montre la figure 5.2. La faible performance d'OpenVZ est une surprise car, comme il s'agit d'un logiciel de virtualisation au niveau du système d'exploitation, nous espérions de sa part une performance proche de celle de Linux natif pour toutes les expériences.

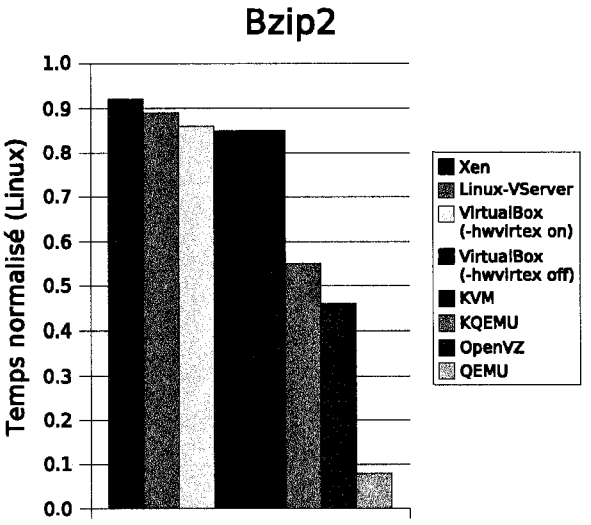

FIG. 5.2: Évaluation des logiciels de virtualisation avec l'utilitaire de compression de fichiers Bzip2

5.1.3 Utilisation de Dbench pour évaluer la performance d'un serveur de fichiers

La figure 5.3 présente les résultats de l'expérience basée sur Dbench, un banc d'essai qui simule la charge de travail d'un serveur de fichiers. Lors de cette expérience, le seul logiciel de virtualisation à présenter une performance similaire à celle de Linux natif est Linux-VServer, la performance des autres logiciels évalués étant de plus de 70% inférieure à celle obtenue par Linux natif. Cette liste inclût Xen qui a montré une meilleure performance lors d'expériences similaires [2, 61]. Malheureusement, la méthodologie employée lors de la réalisation de ces expériences n'a pas été mentionnée dans les publications, ce qui rend difficile une comparaison avec les résultats rencontrés. Linux-VServer est le seul logiciel de virtualisation évalué à utiliser les pilotes d'entrée-sortie du système hôte directement, ce qui explique sa bonne performance.

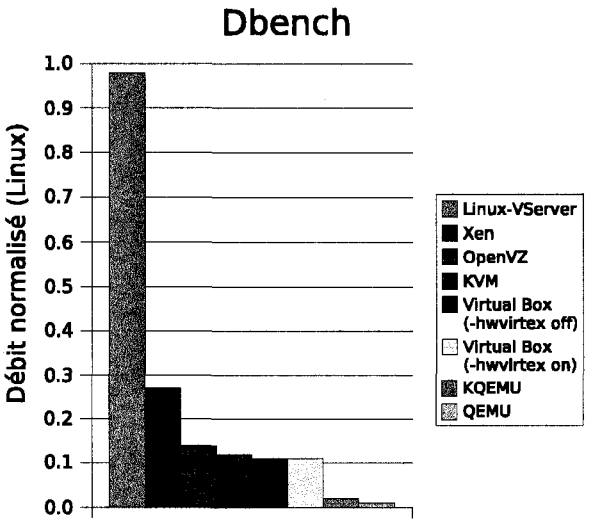

FIG. 5.3: Analyse des logiciels de virtualisation avec Dbench

5.1.4 Utilisation de dd pour évaluer la performance des accès disque

Les résultats pour les expériences de la performance des accès disque réalisées avec le logiciel dd sont présentés à la figure 5.4. Les deux expériences que nous avons menées ne surchargent pas l'UCT mais exigent beaucoup du système d'entrée-sortie.

Lors de la première expérience, une copie d'un fichier contenant une image *iso* de 433 Mo est faite dans la même partition du disque. Pour cette charge de travail, Linux-VServer présente une performance qui dépasse en presque 30% celle de Linux natif. Nous n'avons pas analysé en profondeur les raisons de ce comportement mais il semble probable que les modifications incorporées au noyau de Linux de base par le correctif de VServer procurent un avantage certain à ce type de tâche. Xen et KVM montrent une bonne performance tandis qu'OpenVZ présente une fois de plus une faible performance.

Lors de la deuxième expérience, 60 Go de caractères nuls sont lus du fichier spécial */dev/zero* et écrits sur le périphérique spécial */dev/null*. Comme les données ne sont pas proprement « écrites » sur le disque mais écartées, le rôle principal de cette expérience consiste à faire exécuter des opérations d'entrée-sortie par le système d'exploitation sans charger physiquement le disque. Si on compare les résultats de cette

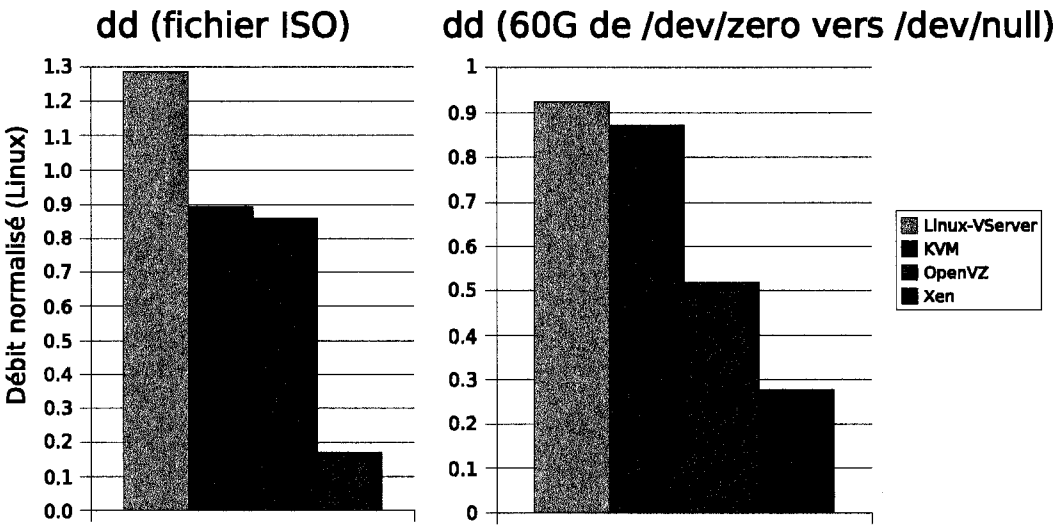

FIG. 5.4: Utilisation de dd pour évaluer la performance de l'accès disque des logiciels de virtualisation

expérience avec ceux obtenus lors de l'expérience précédente, Xen présente une performance moindre tandis que OpenVZ montre une meilleure performance, laquelle demeure toutefois relativement faible si comparée à celle de Linux natif. Les performances de Linux-VServer et de KVM sont, pour cette expérience, proches de celle de Linux natif.

Nous n'avons pas réussi à mesurer correctement les résultats des deux expériences menées avec KQEMU et VirtualBox. En effet, les valeurs retournées par dd pour le temps d'exécution et pour le débit avec ces deux logiciels de virtualisation s'avèrent inexactes lorsqu'on les compare avec l'horloge du système d'exploitation hôte. Il est difficile d'expliquer ce phénomène, toutefois le système se comporte comme si l'exécution de dd ralentissait l'horloge de la machine virtuelle, un incident que nous n'avons pas observé lors des autres expériences.

5.1.5 Utilisation de Netperf pour évaluer la performance réseau

La figure 5.5 présente les résultats pour l'exécution de Netperf, un banc d'essai simple qui utilise un flot de paquets TCP pour évaluer la performance de l'échange

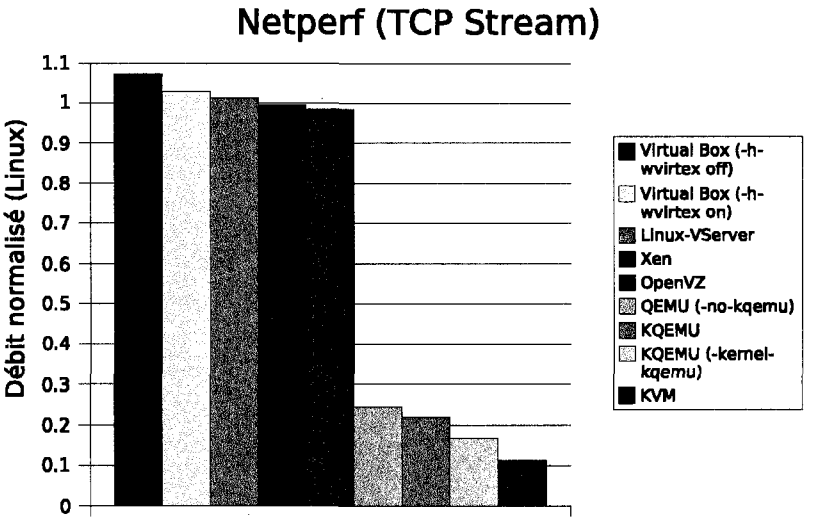

FIG. 5.5: Le banc d'essai Netperf utilise un flot de paquets TCP pour évaluer la performance de l'échange de données par réseau

de données par réseau. Nous pouvons clairement différencier deux groupes dans les résultats de cette expérience : les logiciels de virtualisation qui sont fortement basés sur QEMU, qui ont tous présenté une faible performance, et les autres, qui ont présenté une excellente performance. Il est important de signaler la performance de VirtualBox, laquelle est légèrement supérieure à celle de Linux natif. Finalement, pour la première fois dans notre évaluation, la performance d'OpenVZ est comparable à celle de Linux natif.

5.1.6 Utilisation de Rsync pour évaluer la performance réseau

Nous avons exécuté deux autres expériences pour évaluer la performance du réseau des machines virtuelles en utilisant Rsync pour faire le transfert de données d'un serveur virtuel vers une machine cliente. La première expérience comprend le transfert de l'arbre du noyau de Linux au complet, lequel est composé de 23741 petits fichiers et totalise 294 Mo. La deuxième expérience comprenait le transfert du fichier contenant l'image *iso* utilisée dans l'expérience de compression avec Bzip2. La figure 5.6 présente les résultats des deux expériences. Celles-ci confirment que la force d'OpenVZ semble être dans la réalisation de tâches qui incluent le transfert de données sur le réseau,

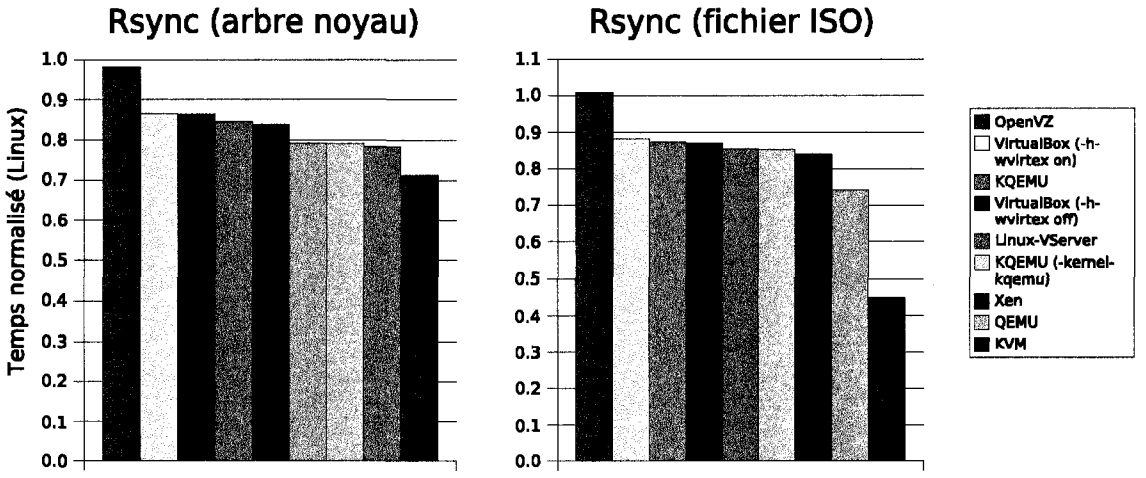

FIG. 5.6: Évaluation de la performance d'échange des données par réseau avec Rsync

un point déjà vérifié avec Netperf dans l'expérience précédente. Elles confirment aussi que la faiblesse de KVM est justement liée aux tâches qui incluent le transfert de données par réseau. Les autres logiciels de virtualisation ont tous présenté une bonne performance pour les deux expériences utilisant Rsync. Même KQEMU, qui n'obtient pas une bonne performance pour Netperf, a montré des bons résultats avec Rsync. Ceci peut être expliqué en fonction de la nature des charges de travail. Une comparaison superficielle entre ces deux bancs d'essai utilisant l'outil *vmstat* pour surveiller leur exécution indique que le nombre de changements de contexte du système durant l'exécution de Netperf est trois fois plus grand que durant l'exécution de Rsync.

5.1.7 Utilisation de Sysbench pour évaluer la performance d'un serveur de BDs

La figure 5.7 présente les résultats de l'expérience utilisant le banc d'essai *OLTP* de la suite Sysbench, conçu pour évaluer la performance d'un serveur de bases de données. Cette charge de travail est composée d'un ensemble de 10000 transactions SQL effectuées sur une base de données MySQL.

Linux-VServer et Xen sont les seuls logiciels de virtualisation à présenter une performance proche de celle de Linux natif pour cette expérience. KVM et OpenVZ

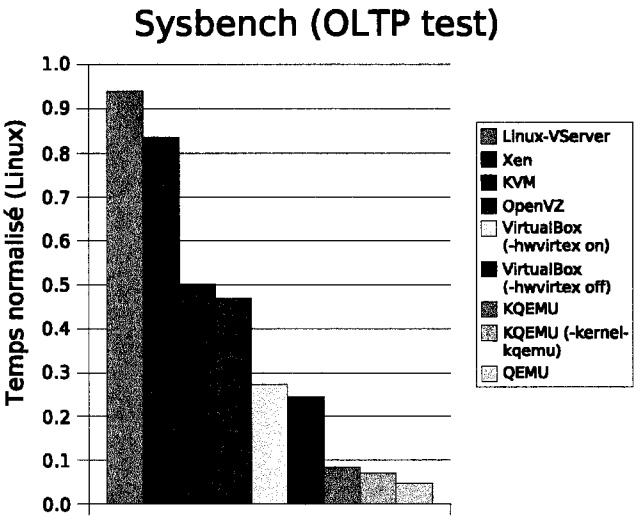

FIG. 5.7: Évaluation de la capacité des logiciels de virtualisation d'accueillir un serveur de base de données avec Sysbench

présentent une performance de près de 50% inférieure à celle de Linux natif. VirtualBox présente une performance de plus de 70% inférieure à celle de Linux natif tandis que la performance KQEMU est de plus de 90% inférieure à la performance présentée par Linux natif.

La charge de travail utilisée dans cette expérience est un bon exemple d'un cas réel d'utilisation d'un serveur. Il est intéressant de voir que, à l'exception de OpenVZ, pour lequel on s'attendait une meilleure performance, les autres logiciels de virtualisation présentent une performance qui correspond à la performance théorique des technologies de virtualisation respectives, illustré par la figure 2.1.

5.2 Évaluation de la capacité de mise à l'échelle

Pour la deuxième partie de notre évaluation, nous avons choisi Sysbench pour évaluer la capacité des différents logiciels de virtualisation à partager les ressources physiques de la machine entre de multiples machines virtuelles qui exécutent la même charge de travail. Cette expérience est répétée six fois avec chaque logiciel de virtualisation en commençant avec une seule machine virtuelle puis en doublant à chaque

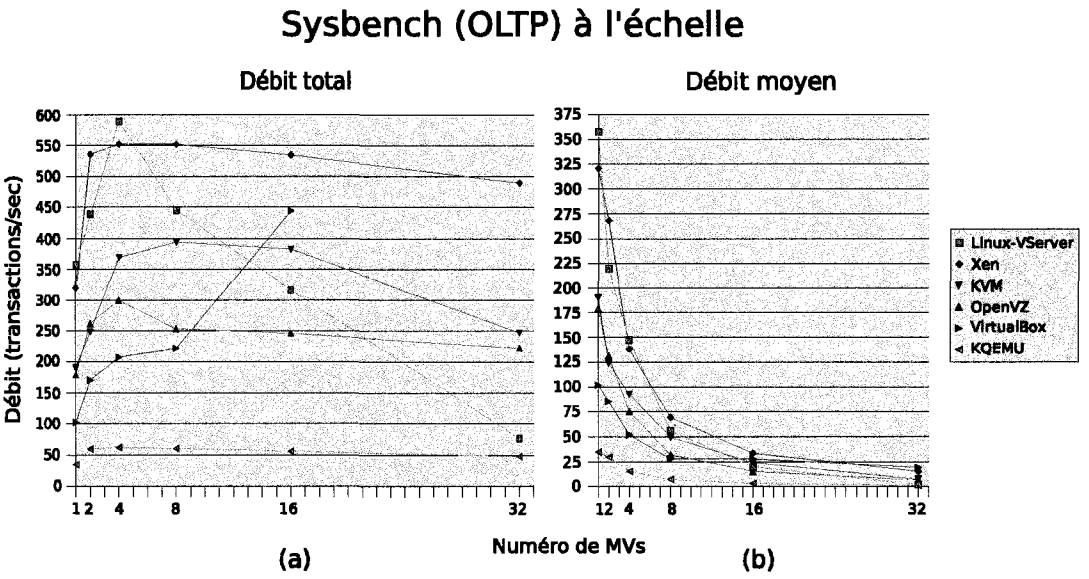

FIG. 5.8: Utilisation de Sysbench pour évaluer la capacité des logiciels de virtualisation de partager les ressources physiques de la machine

itération le nombre de machines virtuelles en exécution.

La figure 5.8 présente les résultats obtenus pour les logiciels de virtualisation lors de chacune des itérations. La figure 5.8(a) présente le débit cumulé par les n machines virtuelles pour chacune des itérations. Ceci est obtenu en multipliant le débit moyen des machines virtuelles à chaque itération, présenté à la figure 5.8(b), par le nombre de machines virtuelles exécutant le banc d'essai de manière concurrente.

Pour tous les logiciels de virtualisation sauf KVM et VirtualBox, le plus grand débit cumulé est obtenu avec l'exécution simultanée des bancs d'essai sur quatre machines virtuelles. Xen et KQEMU ont présenté un débit cumulé presque constant pour les six itérations au cours de l'expérience, mais avec des performances opposées : si Xen peut être considéré comme le logiciel de virtualisation le plus efficient pour cette charge de travail particulière, la faiblesse de KQEMU à gérer de manière efficace les ressources disponibles s'est montrée évidente.

Les deux performances les plus intéressantes ont été présentées par VirtualBox et Linux-VServer. Le débit cumulé présenté par le premier au cours de l'expérience a crû graduellement jusqu'au moment où le nombre de machines virtuelles en exécu-

tion est arrivé à huit. Quand nous avons doublé le nombre de machines virtuelles, passant à 16, le débit moyen par machine virtuelle est resté le même, doublant le débit total produit par l'ensemble de machines virtuelles. Pourtant, nous n'avons pas réussi à exécuter cette expérience avec 32 machines virtuelles, la mémoire disponible par machine virtuelle n'étant pas suffisante pour l'exécution du banc d'essai dans l'environnement virtuel fourni par VirtualBox.

Avec Linux-VServer (ou simplement VServer), le débit cumulé par deux ainsi que par huit machines virtuelles, ou VServers, est pratiquement le même, pourtant il baisse considérablement quand on exécute le banc d'essai dans 16 et 32 VServers en même temps. Ceci n'est pas le comportement que l'on s'attend à rencontrer dans des systèmes en production utilisant ce logiciel de virtualisation. À la recherche d'une explication, nous avons contacté Marc Fiuczynski et Herbert Pötzl, le dernier étant le développeur-chef du projet Linux-VServer. Ils nous ont donné des pistes qui pouvaient aider à expliquer la performance de VServer lorsque cette charge de travail particulière est exécutée à l'échelle. En résumé, ils nous ont suggéré de répéter l'expérience en utilisant différents ordonnanceurs (de l'anglais *scheduler*) d'entrée-sortie du noyau de Linux (`CONFIG_DEFAULT_IOSCHED`) ainsi que différentes fréquences pour le temporisateur (de l'anglais *timer*) du noyau (`CONFIG_HZ`). Ils nous ont aussi suggéré d'exécuter l'expérience de mise à l'échelle directement sur le système d'exploitation hôte, avec et sans le correctif de VServer appliqué sur le noyau, et de comparer les résultats obtenus avec ceux de VServer. La figure 5.9 présente les résultats de cette nouvelle série d'expériences. Sur la légende, les informations entre parenthèses sont, respectivement, la version d'Ubuntu, la fréquence du temporisateur et l'ordonnanceur d'entrée-sortie utilisés par chaque expérience.

Le correctif de VServer utilise la valeur par défaut de la fréquence du temporisateur du noyau de Linux, configuré à 100 Hz, ainsi que l'ordonnanceur d'entrée-sortie standard du noyau de Linux, appelé *deadline*. Xen utilise une configuration différente : la fréquence du temporisateur est configurée à 250 Hz et l'ordonnanceur d'entrée-sortie choisi est *«cfq»*. Nous avons répété l'expérience avec 32 VServers exécutant Sysbench de manière concurrente, et utilisant différentes combinaisons de fréquence pour le temporisateur et d'ordonnanceurs d'entrée-sortie. Nous avons aussi répété l'expérience en utilisant différentes partitions LVM par VServer à la place d'une seule partition par-

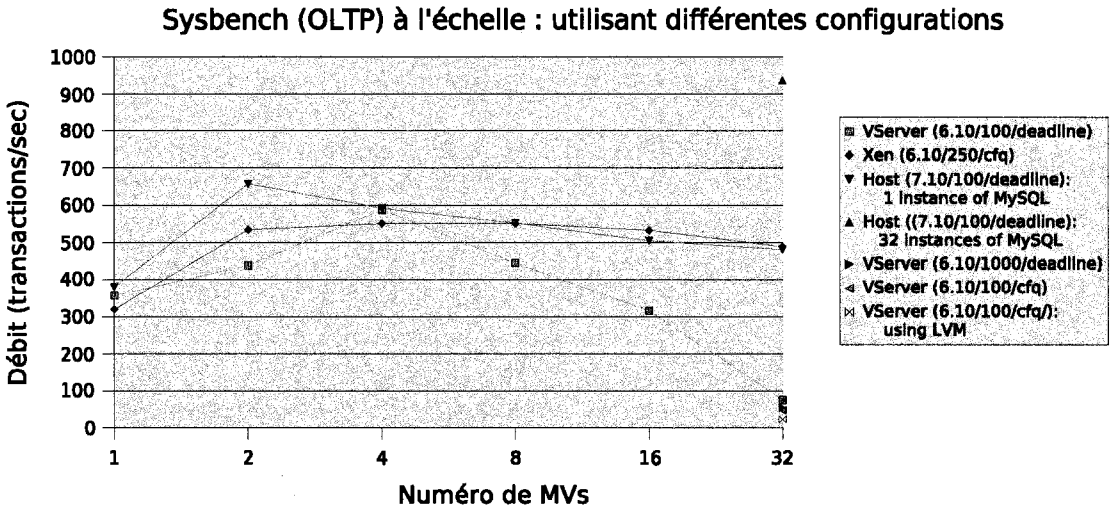

FIG. 5.9: Répétition de l'évaluation de mise à l'échelle de Linux-VServer avec différentes configurations

tagée par tous les VServers. Aucune de ces configurations n'a produit de meilleurs résultats que ceux obtenus par l'expérience originale avec VServer, comme le montre la figure 5.9, ce qui démontre que ni le choix de la fréquence du temporisateur ni le choix de l'odonnanceur d'entrée-sortie ni l'utilisation d'une seule partition partagée par toutes les machines virtuelles n'est responsable de la mauvaise performance de VServer.

Étant informé de ces résultats, Pötzl a suggéré la possibilité que le problème soit relié au noyau de Linux directement et pas au correctif de VServer. Pour analyser cette possibilité, nous avons répété l'expérience en exécutant 1, 2, 4, 8, 16 et finalement 32 instances de Sysbench directement sur le système d'exploitation hôte, toutes les instances accédant à la même base de données. La courbe de performance résultante ressemble à celle de Xen. Le débit cumulé diminue doucement quand il y a plus que quatre instances en exécution en même temps et ne suit pas le patron présenté par VServer. Nous avons aussi répété cette expérience en exécutant 32 instances de Sysbench sur le système d'exploitation hôte, chacune connectée à des bases de données différentes, hébergées par 32 serveurs *mysqld* exécutés sur différents ports du système d'exploitation. Cette configuration a obtenu le plus grand débit cumulé entre toutes

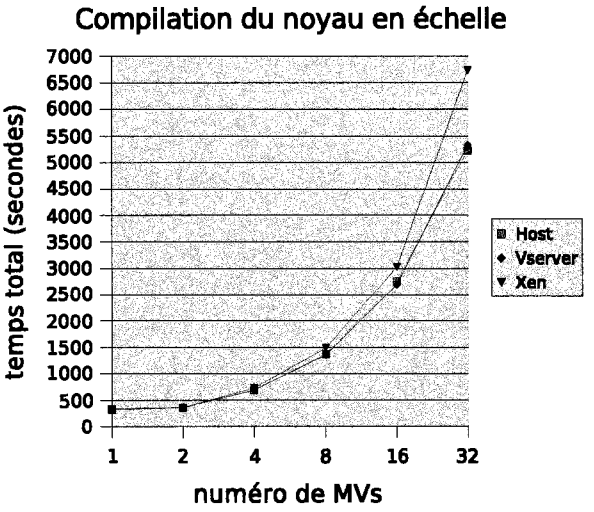

FIG. 5.10: Évaluation de la capacité de mise à l'échelle de Linux-VServer et Xen utilisant la compilation du noyau de Linux comme paramètre

les expériences réalisées. Ces deux dernières expériences, pour lesquelles les résultats sont aussi illustrés par la figure 5.9, confirment que le problème de performance de VServer ne provient pas du noyau de Linux.

Le projet *PlanetLab* [47] utilise le logiciel de virtualisation Linux-VServer pour partager l'infrastructure de leurs grappes de machines en « tranches » (appelées *slices* en anglais). En continuant notre recherche pour la solution du problème, nous avons testé un correctif utilisé par ce projet, lequel règle un problème de fonctionnement de l'ordonnanceur d'UCT présent dans la version de Linux-VServer utilisée dans nos expériences, sans toutefois obtenir de meilleurs résultats.

Finalement, nous avons décidé d'essayer une charge de travail différente : la compilation du noyau de Linux exécutée à l'échelle. Pour les expériences réalisées avec Linux-VServer et Xen nous avons utilisé les mêmes configurations que lors des premières expériences de mise à l'échelle. Pour l'expérience avec le système d'exploitation hôte nous avons utilisé différents répertoires contenant chacun une copie de la source du noyau de Linux. La figure 5.10 présente les résultats de cette expérience, montrant le temps cumulé dans chaque série.

Pour cette charge de travail en particulier, Linux-VServer présente une perfor-

mance équivalente à celle de Linux natif. La performance de Xen est bonne pourtant il présente une légère perte d'efficience lorsqu'il est comparée à Linux natif et Linux-VServer. La sous-performance présentée par le logiciel devient plus significative quand l'expérience est faite avec 32 machines virtuelles.

Les résultats de cette dernière expérience aident à montrer comment la nature de la charge de travail interfère avec la performance des solutions de virtualisation, surtout lorsqu'elles sont exécutées à l'échelle. Ils montrent aussi que Linux-VServer peut livrer la performance qu'on attend de ce logiciel de virtualisation. Nous n'avons pu identifier la source du problème de performance de Linux-VServer lors de l'exécution de Sybench à l'échelle. Nous avons suggéré à l'équipe de développement de ce logiciel de virtualisation d'essayer de réproduire cette expérience. Nous espérons qu'il donnerons éventuellement suite à notre suggestion.

Conclusion

Nous avons réalisé une étude sur la virtualisation de serveurs, avec un accent sur la virtualisation de serveurs x86 utilisant le système d'exploitation Linux et des logiciels de virtualisation libres. Nous avons présenté l'origine de la virtualisation dans les années 1960 et son évolution marquée par les adaptations apportées à l'architecture x86. Le projet original de cette architecture ne permettait pas l'implémentation de la virtualisation telle que définie dans la littérature, aujourd'hui appelée « virtualisation classique ». Pourtant, le besoin de rendre cette architecture virtualisable se montrait de plus en plus nécessaire. Au cours des années, différentes techniques ont été développées pour contourner les limites de l'architecture x86 de manière à permettre sa virtualisation. Nous avons regroupé ces différentes techniques en trois catégories, ou *technologies* de virtualisation, en fonction de leurs caractéristiques communes : virtualisation complète, paravirtualisation et virtualisation au niveau du système de exploitation. Récemment, avec l'ajout de nouvelles extensions à l'architecture x86 par les deux principaux fabricants de micro-processeurs, Intel et AMD, il est maintenant possible d'implanter la virtualisation classique sur des ordinateurs équipés des nouvelles puces.

L'objectif de notre travail était de faire une étude comparative de l'efficience des six logiciels libres de virtualisation suivants : *KQEMU*, *KVM*, *Linux*-VServer, *OpenVZ*, *Xen* et *VirtualBox*. L'efficience des logiciels de virtualisation est évaluée en fonction de la perte de performance qu'ils engendrent ainsi que de leur capacité de mise à l'échelle. Pour ce faire, nous avons divisé notre évaluation en deux parties. Dans la première partie, nous avons utilisé sept bancs d'essai pour évaluer la performance des logiciels de virtualisation et la comparer avec celle de Linux exécuté dans un environnement non virtualisé (Linux *natif*), observant le surcoût imposée par la couche de virtualisation.

Ensuite, nous avons évalué la capacité de mise à l'échelle des logiciels de virtualisation avec l'exécution concurrente du même banc d'essai dans n machines virtuelles, où n=1, 2, 4, 8, 16 et 32.

Pour la première partie de l'évaluation, la performance de Linux-VServer s'est montrée proche de celle de Linux natif, présentant une perte d'efficience de faible à non existant dans toutes les situations sauf une, pour laquelle la performance a surpassé celle présentée par Linux natif pour des opérations d'accès au disque avec la commande *dd*. Nous croyons que les modifications apportées au noyau de Linux par le correctif de Linux-VServer facilite l'exécution de ce type de tâche. Pourtant, pour l'expérience de mise à l'échelle utilisant le banc d'essai *Sysbench*, Linux-VServer ne présente pas le débit cumulé attendu et ce spécialement quand il y a plus de quatre machines virtuelles en exécution. Une analyse plus détaillée a montré que le problème n'est pas lié au noyau de Linux. Nous avons essayé, pour le noyau, différentes combinaisons de planificateur d'entrée-sortie ainsi que différentes fréquences d'horloge interne, sans obtenir de meilleurs résultats. Nous avons décidé de répéter l'expérience de mise à l'échelle avec Linux natif, Linux-VServer ainsi que Xen en utilisant une charge de travail différente : la compilation du noyau de Linux. Cette fois la performance de Linux-VServer s'est montrée comparable à celle de Linux natif, tandis que Xen a présenté une légère perte d'efficience au départ qui est devenue plus significative lorsque le nombre de machines virtuelles en exécution est passé à 32.

La performance de Xen s'est montrée assez bonne dans toutes les expériences avec l'exception de celle utilisant le banc d'essai pour le système de fichiers *Dbench*. En effet, aucun logiciel de virtualisation, à part Linux-VServer, n'obtient de bons résultats avec ce banc d'essai. Xen a aussi été le logiciel de virtualisation à présenter le meilleur débit cumulé dans l'expérience de mise à l'échelle utilisant Sysbench.

KVM obtient, de façon générale, de bonnes performances, spécialement pour un logiciel de virtualisation classique. Nos résultats indiquent qu'il est un bon candidat comme outil de développement mais qu'il doit être évité pour l'exécution d'applications faisant une utilisation intensive du réseau.

La performance d'OpenVZ s'avère moins bonne que prévue pour un logiciel de virtualisation au niveau du système d'exploitation, l'exception étant pour la compilation du noyau de Linux ainsi que les charges de travail faisant appel à l'échange

de données sur le réseau. Le transfert de données par le réseau semble être le point fort de ce logiciel de virtualisation. VirtualBox montre aussi de bons résultats pour les expériences utilisant le réseau mais n'obtient pas de bons résultats pour les autres expériences, à l'exception de celle portant sur la compression de fichier avec *Bzip2*. Finalement, KQEMU s'est montré un bon candidat comme outil de développement mais son utilisation dans des systèmes en production doit être évitée à cause de sa faible performance généralisée ainsi que de sa faible capacité de mise à l'échelle.

Pour la consolidation de serveurs Linux, les technologies de virtualisation telles que la paravirtualisation et la virtualisation au niveau du système d'exploitation semblent faire une meilleure utilisation des ressources physiques disponibles. Pourtant, nos résultats indiquent que la capacité de mise à l'échelle des logiciels de virtualisation est fortement liée à la nature de la charge de travail exécutée. À l'exception de Linux-VServer et de Xen, pour lesquels nous avons testé la compilation du noyau de Linux à l'échelle en plus de Sysbench, nous n'avons pas utilisé des charges de travail différentes pour évaluer la capacité de mise à l'échelle des logiciels de virtualisation. Nous suggérons que cela soit pris en considération dans de futurs travaux.

Il pourrait être intéressant de répéter les expériences de mise à l'échelle en utilisant simultanément une combinaison de différentes charges de travail. À l'exception des centres hébergeant des sites web, l'exécution de plusieurs applications exécutant la même charge de travail sur un même serveur physique n'est pas très répandue.

Finalement, nous suggérons d'installer dans les machines virtuelles la même version du système d'exploitation utilisée par l'hôte. Nous n'avons pas procédé de cette façon dans notre étude et cela a affaibli, quoique de façon minimale, la comparaison des résultats obtenus, en fonction de différentes versions du noyau de Linux et des utilitaires inclus dans chaque version du système d'exploitation choisi.

Annexe A

Configurations des expériences

Un aspect important des études comparatives est de s'assurer que les expériences ont été réalisées de façon juste et contrôlée. Cette section contient l'information détaillée qui permet de réproduire les expériences réalisées dans cette étude.

A.1 Exécution des bancs d'essai

Cette section présente les commandes utilisées pour la préparation ainsi que pour l'exécution des bancs d'essai.

A.1.1 Compilation noyau de Linux

- Préparation

```
$ tar linux-2.6.22.14.tar.gz
$ cd /opt/linux-2.6.22.14
$ make defconfig
```

- Exécution

```
$ date +%s.%N && make && date +%s.%N
$ make clean
```

A.1.2 Dbench

- Préparation

```
$ tar zxvf dbench-3.04.tar.gz
$ cd dbench-3.04
$ ./configure
$ make
$ make install
```

- Exécution

```
$ /usr/local/bin/dbench -t 300 -D /var/tmp 100
```

A.1.3 Netperf

- Préparation

```
$ tar zxvf netperf-2.4.4.tar.gz
$ cd netperf-2.4.4
$ ./configure
$ make
$ make install
```

- Exécution sur le serveur

```
$ netserver
```

- Exécution sur la machine cliente

```
$ netperf -H <server>
```

A.1.4 Rsync

- Fichier /etc/rsyncd.conf utilisé par le démon *rsyncd*

```
uid = root
gid = root
use chroot = no
max connections = 1
```

```
syslog facility = local5
pid file = /var/run/rsyncd.pid
read only = yes
ignore errors = true
ignore nonreadable = true
list = false
strict modes = true
[kernel]
path = /opt/linux-2.6.22.14  # 294M
[iso]
path = /opt/iso              # 433M
```

- Expérience 1 (exécution sur la machine cliente)

```
$ date +%s.%N && rsync -av <server> : :kernel /var/tmp && \
> date +%s.%N
$ rm -fr /var/tmp/*
```

- Expérience 2 (exécution sur la machine cliente)

```
$ date +%s.%N && rsync -av <server> : :iso /var/tmp && date \
> +%s.%N
$ rm -fr /var/tmp/*
```

A.1.5 dd

- Expérience 1

```
$ dd if=/opt/iso/ubuntu-6.06.1-server-i386.iso of=/var/tmp/out.iso
$ rm -fr /var/tmp/*
```

- Expérience 2

```
$ dd if=/dev/zero of=/dev/null count=117187560  # 117187560 = 60G
```

A.1.6 Bzip2

```
$ cp /opt/ubuntu-6.06.1-server-i386.iso .
$ date +%s.%N && bzip2 -9 ubuntu-6.06.1-server-i386.iso && date \
> +%s.%N
$ rm ubuntu-6.06.1-server-i386.iso.bz2
```

A.1.7 Sysbench

L'exécution de Sysbench est divisée en trois étapes : la création de la base de données « sbtest », la préparation du banc d'essai *OLTP* et l'exécution du banc d'essai.

- Préparation

```
$ apt-get install mysql-server libmysql++-dev
$ tar zxvf sysbench-0.4.8.tar.gz
$ cd sysbench-0.4.8
$ ./configure --with-mysql
$ make
$ make install
```

- Création de la base de données « sbtest »

```
$ mysql
$ mysql> create database sbtest ;
$ mysql> exit
```

- Exécution de la phase préparatoire du banc d'essai *OLTP*

```
$ sysbench --test=oltp --mysql-user=root --mysql-host=localhost \
> --debug=off prepare
```

- Exécution du banc d'essai

```
$ sysbench --test=oltp --mysql-user=root --mysql-host=localhost \
> --debug=off run
```

A.2 Configuration des logiciels de virtualisation

Cette section présente les configurations utilisées avec chaque logiciel de virtualisation évalué. Elle n'est pas exhaustive et ne contient que les étapes jugées nécessaires à la reproduction des expériences.

A.2.1 Linux-VServer

- Compilation du noyau VServer

```
$ cd /usr/src
$ cp /home/fernando/Benchmarks/linux-2.6.22.14.tar.bz2 .
$ cp /home/fernando/Benchmarks/patch-2.6.22.10-vs2.2.0.5.diff .
$ bunzip2 linux-2.6.22.14.tar.bz2
$ tar xvf linux-2.6.22.14.tar
$ cd linux-2.6.22.14/
$ cat ../patch-2.6.22.10-vs2.2.0.5.diff |patch -p1
$ make oldconfig
$ make menuconfig
$ make
$ make install
$ make modules_install
$ mkinitramfs -o /boot/initrd.img-2.6.22.14 2.6.22.14
```

- Modification du fichier /boot/grub/menu.lst

```
title kernel Linux-Vserver 2.6.22-14
root (hd0,1)
kernel /boot/vmlinuz-2.6.22.14
       root=UUID=7cb9a13c-2061-4c98-a8da-e5edfcf0ee10 ro
initrd /boot/initrd.img-2.6.22.14
```

- Utilisation du logiciel *vsmanage* pour créer les VServers

```
$ vsmanage create vserver1 10.145.1.11 --image=ubuntu-6.06-i386
```

A.2.2 Xen

- Installation du noyau Xen et des outils de gestion

```
$ apt-get install xen-hypervisor-3.1 xen-tools xen-utils-3.1 \
> linux-headers-2.6.22-14-xen linux-image-2.6.22-14-xen
```

- Correction de bogues avec le script /etc/xen/scripts/network-bridge

```
-vifnum=${vifnum :-$(ip route list | awk '/^default /
  { print $NF }' | sed 's/^[^0-9]*//')}
+vifnum=${vifnum :-$(netstat -arn | awk '/^0.0.0.0 /
  { print $NF }' | sed 's/^[^0-9]*//')}
```

- Création d'une image Xen utilisant comme source le Gulus

```
$ xen-create-image --hostname=xen1 --dir=/media/vms \
> --size=1Gb --swap=512Mb --ip=10.145.1.81 \
> --netmask 255.255.255.0 --gateway=10.145.1.1 --memory=2Gb \
> --install-method=debootstrap --dist=dapper --fs=ext3 --force
> --mirror=http ://gulus.usherbrooke.ca/ubuntu/ \
> --mac=00 :16 :3E :11 :12 :22
```

A.2.3 KVM

- Installation du module *kvm*

```
$ apt-get install kvm qemu
$ modprobe kvm-intel
```

- Création d'une image QEMU utilisée par KVM ainsi que par KQEMU

```
$ dd if=/dev/zero of=/media/vms/disk.img bs=1G count=4
$ qemu-img create /media/vms/disk.img -f qcow2 4G
```

- Démarrage de l'installation du SE invité dans la machine virtuelle

```
$ kvm -hda /media/vms/disk.img -cdrom  \
> /root/ubuntu-6.06.1-server-amd64.iso -m 1024 -boot d
```

- Création d'un pont réseau (/etc/network/interfaces)

```
# The loopback network interface
auto lo
iface lo inet loopback
# Bridge
auto br0
iface br0 inet static
address 192.168.1.3
netmask 255.255.255.0
network 192.168.1.0
bridge_ports eth1
bridge_stp off
bridge_maxwait 5
up /sbin/ifconfig eth1 inet 0.0.0.0 promisc
# Reseau prive
auto eth1
iface eth1 inet static
address 192.168.1.1
network 192.168.1.0
netmask 255.255.255.0
# Reseau public
auto eth0
iface eth0 inet static
address 10.145.1.80
network 10.145.1.0
netmask 255.255.255.0
broadcast 10.145.1.255
gateway 10.145.1.1
```

- Modification du script /etc/kvm/kvm-ifup

```
- /usr/sbin/brctl addif ${switch} $1
+ /usr/sbin/brctl addif br0 $1
```

- Démarrage en utilisant une console série (/boot/grub/menu.lst) ...

```
+ serial unit=0 -speed=9600 -word=8 -parity=no -stop=1
+ terminal serial
...
+ ... console=tty0 console=ttyS0,9600n8
```

- ... puis éditer /etc/inittab et ajouter à la fin

```
co :2345 :respawn :/sbin/getty -L ttyS0 9600 vt100
```

- Démarrage de la machine virtuelle en mode *terminal*

```
$ kvm -hda /media/vms/disk.img -m 512 -net nic -net tap \
> -boot c -nographic
```

A.2.4 OpenVZ

- Compilation du noyau OpenVZ

```
$ bunzip2 linux-2.6.22.tar.bz2
$ tar xvf linux-2.6.22.tar
$ gunzip patch-ovz005.1-combined.gz
$ cd linux-2.6.22
$ cp ../kernel-2.6.22-x86_64.config.ovz .config
$ cat ../patch-ovz005.1-combined |patch -p1
$ make
$ make install
$ make modules_install
$ mkinitramfs -o /boot/initrd.img-2.6.22-ovz005 2.6.22-ovz005
```

- Modification du fichier /boot/grub/menu.lst

```
title           OpenVZ, kernel 2.6.22
root            (hd0,1)
kernel          /boot/vmlinuz-2.6.22-ovz005
                root=UUID=03893821-3c3d-4bc3-b5fb-db3be666a8c1 ro
initrd          /boot/initrd.img-2.6.22-ovz005
```

- Correction du bogue réseau

```
if "cat /proc/sys/net/ipv4/ip_forward" =! 1 :
    net.ipv4.conf.all.forwarding=1 --> /etc/sysctl.conf
```

- Installation des outils de gestion

```
$ deb http ://debian.systs.org/ stable openvz >> \
> /etc/apt/sources.list
$ apt-get install vzctl vzquota
```

- Création d'une machine virtuelle OpenVZ

```
$ vzctl create 101 --ostemplate ubuntu-6.06-i386-minimal \
> --ipadd 10.145.1.11 --hostname vz1
```

A.2.5 KQEMU

- Installation du module *kqemu*

```
$ apt-get install qemu libsdl1.2-dev
$ cd /usr/src
$ tar xvf kqemu.tar
$ cd modules
$ ./configure
$ make
$ make install
$ modprobe kqemu
```

- Démarrage de l'image crée avec QEMU

```
$ qemu-system-x86_64 -hda /media/vms/disk.img -m 1536 \
> -boot c -net nic -net tap -nographic
```

A.2.6 VirtualBox

- Compilation de la version *open source* de VirtualBox et chargement du module *vboxdrv*

```
$ bunzip2 VirtualBox-1.5.4_OSE.tar.bz2
$ tar xvf VirtualBox-1.5.4_OSE.tar
$ cd VirtualBox-1.5.4_OSE
$ ./configure
$ source /usr/src/VirtualBox-1.5.4_OSE/env.sh
$ kmk all
# Compilatation du module
$ cd ./out/linux.amd64/release/bin/src
$ make
$ make install
$ modprobe vboxdrv
```

- Création du groupe VirtualBox

```
$ groupadd vboxusers
$ adduser fernando vboxusers
$ chown root :vboxusers /dev/vboxdrv
```

- Configuration du réseau

```
# Installation paquêts pour le bridge
$ apt-get install bridge-utils
sudo chmod 0666 /dev/net/tun
```

- Création du dispositif *tun0* et association avec l'interface *br0*

```
$ sudo /usr/src/VirtualBox-1.5.4_OSE/out/linux.amd64 \
> /release/bin/VBoxTunctl -t vbox0 -u fernando
$ sudo /sbin/ifconfig vbox0 0.0.0.0 promisc up
$ sudo brctl addif br0 vbox0
```

- Démarrage de l'interface graphique utilisée pour la création des images

```
$ LD_LIBRARY_PATH=/usr/src/VirtualBox-1.5.4_OSE/out/ \
> linux.amd64/release/bin/./usr/src/VirtualBox-1.5.4_OSE \
> /out/linux.amd64/release/bin/./VirtualBox
```

- Il n'est pas possible de démarrer une installation standard Ubuntu 6.06 car le noyau standard est SMP et non compatible avec la machine virtuelle VirtualBox.

Il est nécessaire de démarrer la MV en « rescue mode » une première fois et installer le noyau « Complete Linux Kernel ».

```
$ apt-get install linux-686
```

- Ensuite, il faut démarrer la MV avec ce noyau (2.6.15-53) de manière à pouvoir compiler le noyau 2.6.22-14, n'oubliant pas d'ajouter le support pour le module *pcnet32*, nécessaire pour le fonctionnement du réseau.
- Correction de problème de démarrage des images générées. Enlever "modprobe ide-generic" de `/usr/share/initramfs-tools/scripts/local-top/udev` puis

```
$ update-initramfs -u
```

- Exemple de la procédure pour faire une copie d'une machine virtuelle VirtualBox en ligne de commande

```
$ VBoxManage clonevdi /media/vms/ubuntu.vdi /media/vms/ubuntu2.vdi
$ VBoxManage modifyvdi /media/vms/ubuntu2.vdi compact
$ VBoxManage createvm -name ubuntu2 -basefolder /media/vms/ \
> -register
$ VBoxManage modifyvm ubuntu2 -memory 512 -hwvirtex off -nic1 \
> hostif -cableconnected1 on -hostifdev1 vbox0 -macaddress1 auto \
> -hda /media/vms/ubuntu2.vdi -ostype linux26
$ VBoxManage startvm ubuntu2
```

Bibliographie

[1] Keith ADAMS et Ole AGESEN. « A comparison of software and hardware techniques for x86 virtualization ». Dans *ASPLOS-XII : Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA, 2006. ACM Press.

[2] Paul BARHAM, Boris DRAGOVIC, Keir FRASER, Steven HAND, Tim HARRIS, Alex HO, Rolf NEUGEBAUER, Ian PRATT et Andrew WARFIELD. « Xen and the art of virtualization ». Dans *SOSP '03 : Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[3] Alain BEAULIEU. « Moins de machines, mais plus de serveurs ». *Direction Informatique*, 20(3) :31, avril 2007.

[4] Fabrice BELLARD. « QEMU, a fast and portable dynamic translator ». Dans *ATEC '05 : Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[5] Fabrice BELLARD. « QEMU Internals », June 2007. `http://fabrice.bellard.free.fr/qemu/qemu-tech.html`.

[6] Thomas BITTMAN. « Predicts 2004 : Server Virtualization Evolves Rapidly ». Gartner RAS Core Strategic Planning SPA-21-5502, Gartner Inc., November 2003. `https://h30046.www3.hp.com/campaigns/2005/promo/1-SAHP_CSL_2/previews/Predicts_2004_Sever_Virtualization_Evolves_Rapidly.pdf`.

[7] Lucas BONNET. « État de l'art des solutions libres de virtualisation pour une petite entreprise ». Rapport Technique, Bearstech, Decembre 2007. `http://bearstech.com/files/LB-virtualisationEntrepriseBearstech.pdf`.

[8] Matt BRUDZYNSKI. « Small Companies Go All or Nothing on Server Virtualization ». Rapport Technique, Info-Tech Research Group, October 2006. http://www.infotech.com/Home/Press%20Releases/Small%20Companies%20Go%20All%20or%20Nothing%20on%20Server%20Virtualization,%20Info-Tech%20Research%20Group%20Report%20Says.aspx.

[9] Jeffrey P. CASAZZA, Michael GREENFIELD et Kan SHI. « Redefining Server Performance Characterization for Virtualization Benchmarking ». *Intel Technology Journal*, 10(3) :243–252, 2006.

[10] Standard Performance Evaluation CORPORATION. « SPEC Virtualization Committee », Novembre 2007. http://www.spec.org/specvirtualization.

[11] Robert J. CREASY. « The Origin of the VM/370 Time-Sharing System ». *IBM Journal of Research and Development*, 25(5) :483–490, 1981.

[12] Simon CROSBY et David BROWN. « The virtualization reality ». *Queue*, 4(10) :34–41, 2007.

[13] Peter J. DENNING. « ACM president's letter : performance analysis : experimental computer science as its best ». *Commun. ACM*, 24(11) :725–727, 1981.

[14] Benoit des LIGNERIS. « Virtualization of Linux Based Computers : The Linux-VServer Project ». Dans *HPCS '05 : Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, pages 340–346, Washington, DC, USA, 2005. IEEE Computer Society.

[15] Justin M. FORBES. « Why Virtualization Fragmentation Sucks ». Dans *Proceedings of the Linux Symposium*, volume 1, pages 125–129, Ottawa, ON, Canada, June 2007.

[16] Keir FRASER, Steven HAND, Rolf NEUGEBAUER, Ian PRATT, Andrew WARFIELD et Mark WILLIAMSON. « Safe Hardware Access with the Xen Vir tual Machine Monitor ». Dans *Proceedings of the 1st Workshop on Operating System and Architectural Support for On-Demand IT In frastructure*, Boston, MA, USA, October 2004.

[17] Frank E. GILLETT et Galen SCHRECK. « Server Virtualization Goes Mainstream ». Market research, Forrester Research, Inc., February 2006.

[18] Gabriel GIRARD. « Chapitre 5 - Performance ». Notes de cours. Département d'informatique, Université de Sherbrooke. `http://www.usherbrooke.ca/informatique`.

[19] Innotek GMBH. « The VirtualBox architecture », 2008. `http://www.virtualbox.org/wiki/VirtualBox_architecture`.

[20] Cédric Le GOATER, Daniel LEZCANO, Clément CALMELS, Dave HANSEN, Serge E. HALLYN et Hubertus FRANKE. « Making Applications Mobile Under Linux ». Dans *Proceedings of the Linux Symposium*, volume 1, pages 347–368, Ottawa, ON, Canada, July 2006.

[21] Tandem Database GROUP. « NonStop SQL : A Distributed, High-Performance, High-Availability Implementation of SQL ». Dans *Proceedings of the 2nd International Workshop on High Performance Transaction Systems*, pages 60–104, London, UK, 1989. Springer-Verlag.

[22] Irfan HABIB. « Xen ». *Linux Journal*, 2006(145) :4, 2006.

[23] Ryan A. HARPER, Michael D. DAY et Anthony N. LIGUORI. « Using KVM to run Xen guests without Xen ». Dans *Proceedings of the Linux Symposium*, volume 1, pages 179–188, Ottawa, ON, Canada, June 2007.

[24] Risto HAUKIOJA et Neil DUNBAR. « Introduction to Linux Virtualization Solutions ». Rapport Technique, Hewlett-Packard Development Company, L.P., September 2006. `http://opensource.hp.com/techbriefs/haukioja_dunbar.html`.

[25] Gernot HEISER, Volkmar UHLIG et Joshua LEVASSEUR. « Are virtual-machine monitors microkernels done right ? ». *SIGOPS Oper. Syst. Rev.*, 40(1) :95–99, 2006.

[26] IBM. « IBM Systems Virtualization ». Rapport Technique version 2, release 1, International Business Machines Corporation, December 2005. `http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicay/eicay.pdf`.

[27] IBM. « Driving business value with a virtualized infrastructure ». Rapport Technique, International Business Machines Corporation, March 2007. `http://www-935.ibm.com/services/us/cio/optimize/optit_wp_gsw1064_usen-virtual.pdf`.

[28] Xuxian JIANG et Xinyuan WANG. « "Out-of-the-Box" Monitoring of VM-Based High-Interaction Honeypots ». *Recent Advances in Intrusion Detection*, pages 198–218, 2007.

[29] M. Tim JONES. « Virtual Linux », December 2006. `http://www.ibm.com/developerworks/library/l-linuxvirt/index.html`.

[30] M. Tim JONES. « Discover the Linux Kernel Virtual Machine », April 2007. `http://www.ibm.com/developerworks/linux/library/l-linux-kvm/`.

[31] Avi KIVITY, Yaniv KAMAY, Dor LAOR, Uri LUBLIN et Anthony LIGUORI. « kvm : the Linux Virtual Machine Monitor ». Dans *Proceedings of the Linux Symposium*, volume 1, pages 225–230, Ottawa, ON, Canada, June 2007.

[32] Tim KLASSEL et Jeffrey PECK. « The Rise of the Virtual Machine and the Real Impact It Will Have ». Rapport Technique, Thomas Eisel Partners, 2006. `http://www.xensource.com/files/VM_Whitepaper_TWP.pdf`.

[33] Menno LAGEMAN. « Solaris Containers - What They Are and How to Use Them ». Sun blueprints, Sun Microsystems, May 2005. `http://www.sun.com/blueprints/0505/819-2679.pdf`.

[34] Matthieu LAMELOT. « Le point sur la virtualisation », January 2007. `http://www.presence-pc.com/tests/virtualisation-Intel-AMD-512/`.

[35] Kevin LAWTON. « Running multiple operating systems concurrently on an IA32 PC using virtualization techniques », November 1999. `http://www.floobydust.com/virtualization/lawton_1999.txt`.

[36] Vikram MAKHIJA, Bruce HERNDON, Paula SMITH, Lisa RODERICK, Eric ZAMOST et Jennifer ANDERSON. « VMmark : A Scalable Benchmark for Virtualized Systems ». Rapport Technique VMware-TR-2006-002, VMware, Inc., September 2006. `http://www.vmware.com/pdf/vmmark_intro.pdf`.

[37] Shamus MCGILLICUDDY. « Virtualization : SMBs go all the way », November 2006. `http://searchsmb.techtarget.com/originalContent/0,289142,sid44_gci1230805,00.html`.

[38] Larry MCVOY et Carl STAELIN. « LMbench : Portable Tools for Performance Analysis ». Dans *USENIX 1996 Annual Technical Conference, FREENIX Track*, 1996.

[39] Gordon E. MOORE. « Cramming more components onto integrated circuits ». Dans *Readings in computer architecture*, pages 56–59, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[40] Robert MULLINS. « Group to study virtualization benchmarks », October 2006. `http://www.infoworld.com/archives/emailPrint.jsp?R=printThis&A=/article/06/10/31/HNvirtbenchmark_1.html`.

[41] Jun NAKAJIMA et Asit K. MALLICK. « Hybrid-Virtualization : Enhanced Virtualization for Linux ». Dans *Proceedings of the Linux Symposium*, volume 2, pages 87–96, Ottawa, ON, Canada, June 2007.

[42] Susanta NANDA et Tzi cker CHIUEH. « A Survey on Virtualization Technologies ». Rapport Technique, State University of New York, February 2005. `http://www.ecsl.cs.sunysb.edu/tr/TR179.pdf`.

[43] Gil NEIGER, Amy SANTONI, Felix LEUNG, Dion RODGERS et Rich UHLIG. « Intel Virtualization Technology : Hardware Support for Efficient Processor Virtualization ». *Intel Technology Journal*, 10(3) :167–177, 2006.

[44] Office québequois de la langue française. « *Le grand dictionnaire terminologique* », août 2007. `http://www.granddictionnaire.com`.

[45] Tavis ORMANDY. « An Empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments », 2007. Presented at CanSecWest Vancouver 2007. `http://taviso.decsystem.org/virtsec.pdf`.

[46] Pradeep PADALA, Xiaoyun ZHU, Zhikui WANG, Sharad SINGHAL et Kang G. SHIN. « Performance Evaluation of Virtualization Technologies for Server Consolidation ». Rapport Technique HPL-2007-59, HP Laboratories Palo Alto, April 2007. `http://www.hpl.hp.com/techreports/2007/HPL-2007-59.pdf`.

[47] Larry PETERSON et Timothy ROSCOE. « The design principles of PlanetLab ». *SIGOPS Oper. Syst. Rev.*, 40(1) :11–16, 2006.

[48] John PFLUEGER et Sharon HANSON. « Data Center Efficiency in the Scalable Enterprise ». *Dell Power Solutions*, pages 08–14, February 2007.

[49] Pedro PLA. « DRBD in a heartbeat ». *Linux Journal*, 2006(149) :3, 2006.

[50] Gerald J. POPEK et Robert P. GOLDBERG. « Formal requirements for virtualizable third generation architectures ». *Commun. ACM*, 17(7) :412–421, 1974.

[51] Herbert PÖTZL. « Linux-VServer Technology ». `http://www.13thfloor.at/vserver/papers/PAPER.txt`, 2004.

[52] Herbert PÖTZL et Marc E. FIUCZYNSKI. « Linux-VServer : Resource Efficient OS-Level Virtualization ». Dans *Proceedings of the Linux Symposium*, volume 2, pages 151–160, Ottawa, ON, Canada, June 2007.

[53] Benjamin QUETIER, Vincent NERI et Franck CAPPELLO. « Selecting A Virtualization System For Grid/P2P Large Scale Emulation ». Dans *Proc of the Workshop on Experimental Grid testbeds for the assessment of large-scale distributed applications and tools (EXPGRID'06)*, Paris, France, June 2006.

[54] Benjamin QUÉTIER et Vincent NÉRI. « V-Meter : Microbenchmark pour évaluer les utilitaires de virtualisation dans la perspective de systèmes d'émulation à grande échelle ». Dans *16ème Rencontres Francophones du Parallélisme (RenPar'16)*, Le Croisic, France, Avril 2005.

[55] J. ROBIN et C. IRVINE. « Analysis of the Intel Pentium's ability to support a secure virtual machine monitor ». Dans *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[56] Love H. SEAWRIGHT et Richard A. MACKINNON. « VM/370 - A Study of Multiplicity and Usefulness. ». *IBM Systems Journal*, 18(1) :4–17, 1979.

[57] Abraham SILBERSCHATZ, Peter Baer GALVIN et Greg GAGNE. *Operating System Concepts with Java.* John Wiley & Sons, Inc., Hoboken, NJ, USA, 6th édition, 2004.

[58] Amit SINGH. « An Introduction to Virtualization ». `http://www.kernelthread.com/publications/virtualization/`, March 2004.

[59] Equipe Administration Système SMILE. « Livre Blanc Virtualisation ». Rapport Technique, Smile, September 2007. `http://www.smile.fr/content/smile/livreblanc/virtualisation.htm`.

[60] Pam SNAITH. « Preparing for the Reality of Virtualization ». Esm product marketing, CA Inc., January 2007. `http://ca.com/files/WhitePapers/virtualization_wp_en_us.pdf`.

[61] Stephen SOLTESZ, Herbert PÖTZL, Marc E. FIUCZYNSKI, Andy BAVIER et Larry PETERSON. « Container-based Operating System Virtualization : A Scalable,

High-performance Alternative to Hypervisors ». Dans *Proc of the EUROSYS (EUROSYS'07), Lisboa, Portugal, March 21-23*, 2007.

[62] Jon STOKES. « SPEC starts on virtualization benchmark », November 2006. `http://arstechnica.com/news.ars/post/20061113-8210.html`.

[63] Andrew TRIDGELL. « Emulating Netbench ». `http://samba.org/ftp/tridge/dbench/README`.

[64] Rich UHLIG, Gil NEIGER, Dion RODGERS, Amy L. SANTONI, Fernando C. M. MARTINS, Andrew V. ANDERSON, Steven M. BENNETT, Alain KAGI, Felix H. LEUNG et Larry SMITH. « Intel Virtualization Technology ». *Computer*, 38(5) :48–56, 2005.

[65] Melinda VARIAN. « VM and the VM Community : Past, Present, and Future ». SHARE 89, Sessions 9059-9061. `http://www.princeton.edu/~melinda/25paper.pdf`, August 1997.

[66] VMWARE. « A Performance Comparison of Hypervisors ». Rapport Technique, VMware, Inc., 2007. `http://www.vmware.com/pdf/hypervisor_performance.pdf`.

[67] VMWARE. « VMware VMmark Benchmarking Guide ». Rapport Technique VMM-ENG-Q307-248, VMware, Inc., Palo Alto, Ca, USA, August 2007.

[68] Andrew WHITAKER, Marianne SHAW et Steven D. GRIBBLE. « Scale and performance in the Denali isolation kernel ». *SIGOPS Oper. Syst. Rev.*, 36(SI) :195–209, 2002.

[69] WIKIPEDIA. « Operating system-level virtualization », December 2006. `http://en.wikipedia.org/wiki/Operating_system-level_virtualization`.

[70] WIKIPEDIA. « History of CP/CMS », Septembre 2007. `http://en.wikipedia.org/wiki/History_of_CP/CMS`.

[71] WIKIPEDIA. « Hypervisor », Septembre 2007. `http://en.wikipedia.org/wiki/Hypervisor`.

[72] XENSOURCE. « A Performance Comparison of Commercial Hypervisors ». Rapport Technique, XenSource Inc., 2007. `http://www.xensource.com/files/hypervisor_performance_comparison_1_0_5_with_esx-data.pdf`.