Evaluation des performances réseau dans le contexte de la virtualisation XEN

Pascale Primet* — **Olivier Mornard*** — **Jean-Patrick Gelas****

* *INRIA Rhône-Alpes,* ** *Université de Lyon*
Laboratoire LIP
École Normale Supérieure de Lyon
46, allée d'Italie
69364 Lyon Cedex 07 - France
{pascale.primet,olivier.mornard,jpgelas}@ens-lyon.fr

RÉSUMÉ. *Les techniques de virtualisation des systèmes d'exploitation améliorent la sécurité, l'isolation, la fiabilité et la flexibilité des environnements. Ces techniques deviennent incontournables dans les systèmes distribués large échelle et commencent à être exploitées dans les réseaux. Cependant la virtualisation introduit un surcoût devant être intégré aux modèles de performance des dispositifs virtualisés afin d'en prédire le comportement. Dans cet article nous analysons ce surcoût et évaluons expérimentalement l'impact de la virtualisation* XEN *sur les performances réseau dans le cas des réseaux haut débit. Nous montrons que le débit obtenu par TCP est très peu affecté par la virtualisation au prix d'une augmentation de la consommation CPU. Nous étudions les conflits entre le traitement protocolaire et les processus de calcul pour l'accès aux capacités des processeurs. Nous montrons aussi l'impact de l'ajustement de la taille de la fenêtre de congestion de TCP dans le cas de plusieurs machines virtuelles.*

ABSTRACT. *Virtualization technics of the Operating System improve security, isolation, reliability and flexibility of the environments. These technics become a must in large scale distributed system and begin to be used in data transport networks. However, virtualization introduced an overhead which must be integrated to virtualized system performance models in order to forecast their behavior. In this article, we analyse this overhead and give an experimental evaluation of the virtualization impact with* XEN *over the network performance in high speed network context. We show that the throughput reached with TCP is barely affected by the virtulization at the expense of a growing CPU consumption. Moreover, we study the conflicts between protocol process and standard process to access CPU resources. Finally, we show the impact of the TCP's congestion window size in case of several virtual machines running.*

MOTS-CLÉS : *Performances réseaux, virtualisation, Xen*

KEYWORDS: *Network performances, virtualization, Xen*

2e soumission à *CFIP2008*, le 11 février 2008.

1. Introduction

La virtualisation a été proposée par IBM dès le milieu des années 60. L'idée générale est de permettre de lancer de multiples instances de système d'exploitation sur le même matériel. Ainsi chaque instance se comporte fonctionnellement exactement comme si elle était en cours d'exécution seule sur le même matériel. Aujourd'hui, cette approche remporte un immense succès car elle permet d'améliorer la sécurité, l'isolation, la fiabilité et la flexibilité des environnements. Ainsi, ces techniques deviennent incontournables dans les systèmes distribués large échelle dans lesquels ces défis doivent être relevés. Dans le domaine des réseaux, le phénomène est plus récent et plusieurs projets explorent l'intérêt d'utiliser les techniques de virtualisation dans les routeurs eux-mêmes. Les fabriquants de processeurs, quant à eux, fournissent aujourd'hui en standard un jeu d'instructions processeur (virtualisation matérielle) permettant d'isoler efficacement les machines virtuelles, et ce même dans les machines grand public.

Dans le cas des systèmes distribués large échelle, la virtualisation permet la protection contre les programmes malveillants, l'évitement des conflits, la sécurité de l'exécution dans un environnement certifié, la mesure de la consommation de ressources. L'exécution de longs travaux peut être fiabilisée grâce à la possibilité de redémarrer à partir d'un certain état sauvegardé. La virtualisation est très utile quand le redimensionnement de l'environnement en termes de capacités (CPU, mémoire, bande passante d'E/S, etc.) est nécessaire. Enfin la virtualisation permet d'optimiser l'utilisation d'un ensemble de ressources par équilibrage de charge (*load-balancing*) des machines virtuelles sur des machines physiques.

L'utilisation de la virtualisation système dans le cas des routeurs permet de simplifier leur gestion, d'augmenter la sécurité, de protéger certains trafics, d'adapter la gestion des flux en fonction des ressources, de la consommation d'énergie, d'adapter le système et les processus de routage aux trafics et non l'inverse. On peut aussi envisager le partage d'un même équipement par plusieurs organisations. La virtualisation s'avére être aussi une approche très intéressante pour l'expérimentation.

Si la virtualisation offre de nombreux avantages et de nombreuses opportunités, elle apporte néanmoins une contrepartie : son coût non nul et indéterministe en terme de consommation de ressources. Alors que le surcoût et le passage à l'échelle des techniques de virtualisation a été bien étudié au niveau système d'exploitation, peu de travaux ont été menés dans le domaine de l'analyse des performances réseau de bout en bout. Nous nous attachons donc à évaluer et à comprendre le surcoût de consommation de bande passante réseau et de traitement protocolaire, en particulier dans les réseaux haut débit. En effet, ce surcoût doit être intégré aux modèles de performance des dispositifs virtualisés (cluster virtuels ou routeurs virtuels) afin d'en prédire au mieux le comportement. Dans cet article, nous nous focalisons sur l'évaluation de l'environnement *XEN* qui est un logiciel libre très utilisé. La section 2 propose un bref état de l'art des différentes méthodes de virtualisation. Nous développons dans la section 3 une analyse des mécanismes réseau dans le cas de la solution *XEN*. Enfin la

section 4 propose une évaluation des performances réseau obtenues dans le contexte du protocole TCP et de l'environnement *XEN*. La dernière section 6 développe la conclusion et quelques perspectives.

2. Les mécanismes de virtualisation

Avant d'aborder l'étude des interactions entre le réseau et les mécanismes de virtualisation d'OS, nous exposons les différentes approches de virtualisation existantes :

Virtualisation au niveau application : Cette approche permet de virtualiser de multiples serveurs isolés et sécurisés sur un seul serveur physique. Il n'y a pas de système d'exploitation dans les machines virtuelles. Tous les serveurs virtuels partagent le même système d'exploitation : celui de la machine physique. *VServer*[GEL 03] est basé sur cette approche.

Émulateurs : Dans l'approche émulateur, une machine virtuelle est simulée ainsi que tout son matériel spécifique. Il est donc possible d'utiliser un système d'exploitation classique non modifié sur cette architecture. On peut également émuler un processeur différent de celui de la machine hôte. Cette technique est couramment utilisée pour développer des logiciels pour les nouveaux processeurs avant qu'ils ne soient effectivement disponibles.

Système d'exploitation dans l'espace utilisateur : Une autre possibilité est d'exécuter le système d'exploitation dans l'espace utilisateur comme une application classique. Cette solution n'est pas très performante et est principalement utilisée pour les développements des noyaux des systèmes. Le plus célèbre projet basé sur cette approche est *UML* ou *User Mode Linux* [HOX 02].

Paravirtualisation : Par rapport à l'approche précédente, la technique de paravirtualisation ne nécessite pas de simuler le matériel, mais plutôt d'offrir une API offrant les mêmes services que le matériel. Ceci implique une modification du système invité. La série d'abstractions (processeur virtuel, mémoire virtuelle, réseaux virtuels, etc...) permet à différents OS d'être portés. Ces abstractions ne sont pas nécessairement similaires à de réelles ressources matérielles de la machine hôte. Ces services sont fournis par des appels système spéciaux effectués auprès d'un hyperviseur. Le projet le plus connu basé sur cette approche est *XEN* [BAR 03, PRA 05]. *XEN* est un moniteur de machines virtuelles pour l'architecture x86, qui permet des exécutions simultanées de plusieurs OS en fournissant une isolation des ressources et un confinement des différents environnements d'exécution. Dans *XEN*, un domaine 0 (Dom0), créé sur l'ordinateur hôte au démarrage, accède à l'interface de contrôle et exécute l'application de gestion des domaines virtuels. Le domaine 0 permet de contrôler le lancement et l'arrêt des autres domaines dans lesquels les OS invités s'exécutent. *XEN* propose deux ordonnanceurs : «Borrowed Time virtuelle» qui fournit un partage équitable des ressources CPU aux différents domaines, et «Atropos» qui est un ordonnanceur temps réel à contraintes relachées, il autorise un ordonnancement

sur la base du «best effort». Pour la virtualisation de la mémoire, *XEN*réserve pour chaque VM (Virtual Machine) une certaine quantité de mémoire physique. Cette zone mémoire est fixée à l'initialisation de chaque VM, mais elle peut être adaptée si un utilisateur a besoin de plus ou moins de mémoire (sans quitter ou redémarrer la VM). L'utilisation de *XEN* n'est pas transparente, puisque les systèmes d'exploitation invités doivent être portés avant d'être installés pour pouvoir fonctionner. *XEN* fournit un haut degré de confinement, avec la virtualisation du CPU et des ressources.

Virtualisation assistée matériellement : Par rapport à l'approche précédente, la virtualisation assistée par le matériel permet d'utiliser un OS invité non modifié[QUM]. La machine virtuelle dispose de son propre matériel virtuel et les OS peuvent être utilisés isolés les uns des autres. Dans de nombreux cas, la machine virtuelle exécute un système d'exploitation différent de celui de l'ordinateur hôte. Cette approche a été rendue récemment possible (2005) grâce à une amélioration du jeu d'instructions des processeurs fournie par Intel (VT Technology) et AMD (Pacifica), mais aussi par IBM (IBM advanced POWER virtualisation) et Sun (Sun UltraSPARC T1 hypervisor). L'objectif de cette démarche est de faire tourner les OS invités non modifiés sans perte de performances.

3. Virtualisation de l'accés au réseau

3.1. *Protocoles et accès réseau classiques*

Sous Unix, et plus spécifiquement dans sa version Linux, les applications communiquent à travers l'interface *socket* qui permet l'accès aux services de transport (TCP, UDP, DCCP, SCTP,...) implantés par les protocoles de transport du noyau. Ces protocoles interagissent avec les composants du protocole IP qui assurent l'acheminement des paquets à travers les différents équipements intermédiaires jusqu'au destinataire. La couche IP dialogue avec la carte d'interface réseau (typiquement Ethernet) via son pilote (driver ou module) situé dans le noyau et représenté par une "interface". A l'émission, le système copie les données dans une zone spéciale, le socket buffer (`skbuf`), puis rend la main à l'application. Pendant ce temps, le système d'exploitation se charge d'envoyer les données en arrière plan de cette zone vers la carte d'interface réseau par un accès direct à la mémoire (DMA, *Direct Memory Access*). Ainsi l'application n'est bloquée que pendant la première copie, au lieu d'attendre la réception du dernier message d'acquittement. En réception, la stratégie est symétrique, mais le récepteur est vraiment bloqué jusqu'à l'arrivée des données en mémoire utilisateur. Ce modèle a pour inconvénient d'être gourmand en temps processeur et en occupation du bus mémoire (3 accès côté émetteur et 3 côté récepteur). Ainsi, les communications à haut débit consomment un très grande quantité de temps CPU (les traitements protocolaire d'une connexion à 10Gb/s saturent un ordinateur moderne). Le système d'exploitation fournit plusieurs fonctions pour le traitement de la pile protocolaire : l'ordonnancement des tâches liées au protocole, la gestion de la mémoire, la gestion

des interruptions et les opérations d'accès au matériel. L'ordonnancement du protocole a une grande influence sur l'efficacité et les performances temps-réel du protocole.

Au niveau des performances que l'on peut attendre en terme de débit bout en bout entre les deux extrémités d'une connexion Ethernet, deux valeurs peuvent être considérées : le débit émis et le débit utile reçu. Considérons une liaison physique de type Ethernet. Soit R_{EthB} le débit binaire Ethernet. $R_{\text{EthB}} = 10^9 bps$ pour du Gigabit Ethernet. Nous définissons R_{Eth} le débit trame Ethernet en faisant l'hypothèse de trames de 1514 octets (sans le checksum (FCS), le préambule et l'*inter frame gap* (IFG)). $R_{\text{Eth}} = (1514/(1514 + 4 + 12 + 8)) * R_{\text{EthB}} = 984.40$Mbps Nous définissons R_{IP} le débit IP paquets (Ethernet payload). Avec l'hypothèse d'une MTU [1] de 1500 octets : $R_{\text{IP}} = (1500/1514) * R_{\text{Eth}} = 975.30$Mbps.

Le débit d'un protocole de transport TCP idéal sur un lien 1Gbps Ethernet (sans le *slow start* ni le mécanisme de retransmissions de TCP) est défini par R_{TP}. Sous les hypothèses précédentes, $R_{\text{TP}} = ((1500 - 52)/1500) * R_{\text{IP}} = 941.49$Mbps Ainsi cette valeur représente le *goodput* (R_{TP}) et (R_{EthB}) le débit émis effectivement sur le lien. Ainsi le débit utile maximum atteignable sur un lien gigabit Ethernet est donc de 941.49Mbps.

En termes de temps de calcul et d'utilisation processeur, nous proposons le modèle suivant qui considère les différentes étapes de traitement dans le noyau et de l'interface (côté émetteur et côté récepteur). Soient : $C_{\text{driversnd}}(S))$ le coût de transmission, fonction de la taille (S) des données à transmettre ou de réception ($C_{\text{driverrcv}}(S)$) sur la carte (copie), C_{IP_*} le coût du traitement IP en émission ou en réception, C_{TCP_*} le coût de traitement TCP en émission ou en réception, $C_{\text{inet}_*}(S)$ le coût de copie dans/vers l'espace utilisateur (fonction de la taille du message à transmettre), et enfin C_{syscall_*} le coût de l'appel système. Ainsi, au niveau de l'émetteur, nous modélisons le coût CPU $C_{\text{TPsnd}}(S)$ comme suit :

$$C_{\text{TPsnd}}(S) = C_{\text{syscallsnd}} + C_{\text{inetsnd}}(S) + C_{\text{TCPsnd}} + C_{\text{IPsnd}} + C_{\text{driversnd}}(S)$$

Au niveau du récepteur, le coût CPU C_{TPrcv} est :

$$C_{\text{TPrcv}}(S) = C_{\text{driverrcv}}(S) + C_{\text{IPrcv}} + C_{\text{TCPrcv}} + C_{\text{inetrcv}}(S) + C_{\text{syscallrcv}}$$

3.2. *Analyse de l'accès réseau dans XEN*

XEN [BAR 03] ne gère pas les périphériques directement, c'est le travail du noyau Linux exécuté en tant que *domaine 0* (Dom0). Les drivers de toutes les cartes réseaux physiques doivent être fournis par le noyau du Dom0. Chaque machine virtuelle possède une ou plusieurs interfaces réseau virtuelle. La communication entre les interfaces est effectuée en utilisant deux buffers circulaires à jetons (un pour l'envoi, un pour la réception). Afin d'assurer un accès fiable aux matériel ainsi que la possibilité de le paramétrer, la virtualisation du matériel dans *XEN* se conforme au modèle *split*

1. Message Transfer Unit

device driver présenté dans [LES 96]. Le Dom0 peut utiliser un dispositif de routage, de pontage (bridge) ou de NAT virtuel pour transmettre les paquets sur les interfaces physiques. Ainsi, au niveau du réseau, le noyau du domaine invité envoie les paquets IP au pilote frontal qui se charge de les transmettre au pilote générique situé dans le domaine 0 via le module netloop (voir figure 1. Ainsi, avec *XEN*, les paquets sont recopiés du socket buffer vers la zone mémoire associée au driver de la carte réseau du domaine 0. Pour gérer les multiples communications issues des différents domaines invités, *XEN* implémente dans le domaine 0 un routeur, un pont ou même NAT. C'est cet équipement virtuel qui ordonnance les paquets et les transmet sur l'interface physique. Dans le cas du pont virtuel, l'arbitrage entre les différentes communications se fait selon une simple politique FIFO, correspondant à l'ordre d'arrivée des paquets dans le driver du domaine 0.

La figure 2 schématise le principe du bridge logiciel proposé pour l'aiguillage des trames issues des différentes machines virtuelles.

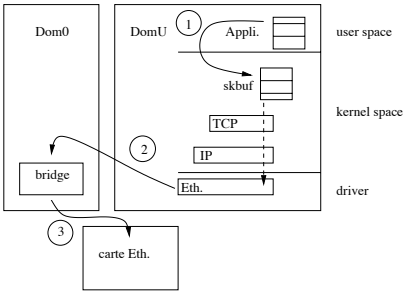

Figure 1. *Le chemin des données et les 3 copies*

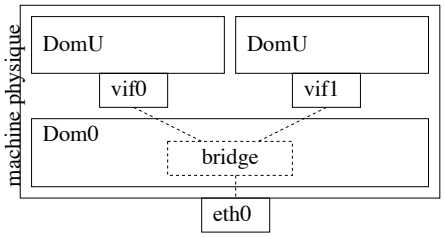

Figure 2. *Bridge logiciel (dans le dom 0)*

Nous proposons donc de compléter le modèle de coût CPU lié aux communications en y intégrant le coût des traitements liés à virtualisation. Au niveau émetteur, nous modélisons C_{TPsndV} :

$$C_{\text{TPsndV}}(S) = C_{\text{TPsnd}}(S) + C_{\text{netloopsnd}}(S) + C_{\text{bridgesnd}} + C_{\text{driversndV}}(S)$$

$$C_{\text{TPsndV}}(S) = C_{\text{syscallsnd}} + C_{\text{inetsnd}}(S) + C_{\text{IPsnd}} + C_{\text{TCPsnd}} + C_{\text{netloopsnd}}(S) + C_{\text{bridgesnd}} + C_{\text{driversndV}}(S)$$

Au niveau du récepteur, le coût CPU C_{TPrcv} est modélisé de manière symétrique. Où $C_{\text{driver*V(S)}}$ correspond au coût de transmission ou de réception sur la carte (copie 1) via le driver générique, $C_{\text{bridge}*}$ au coût de traitement du pont virtuel, $C_{\text{netloop}*(S)}$ au coût de la copie (copie 2) du domaine invité vers le domaine 0 (et réciproquement). On peut s'attendre à ce que la copie entre le domaine invité et le domaine 0 constitue une part importante du coût CPU total de la transmission.

4. Expérimentations et analyse des résultats

L'objectif de ces expérimentations est d'étudier le surcoût de l'implantation de la virtualisation pour l'accès au réseau dans le cas de l'environnement *XEN*. Nous nous concentrons sur les performances de bout en bout obtenues au niveau utilisateur via le protocole TCP. Comme nous l'avons présenté dans la section précédente, la gestion des entrées-sorties réseau dans *XEN* fait intervenir un certain nombre de composants logiciels (en particulier netloop et un équipement virtuel en domaine 0) qui s'intercalent dans le chemin classique des données. Ces composants peuvent être arbitrairement invoqués par l'ordonnanceur des tâches et faire des copies supplémentaires de paquets. Nous avons donc développé un plan d'expériences pour mesurer le coût dû à la virtualisation au niveau du client et au niveau du serveur en termes de débit et de capacité CPU (expérience 1). Le paramétrage de TCP étant relativement délicat dans le cas des réseaux haut débit, nous avons étudié les différentes valeurs de taille de buffer (expérience 2). Enfin, comme la virtualisation permet d'instancier plusieurs machines virtuelles sur une même machine, nous avons cherché à savoir si le partage des ressources était équitable et prévisible (expérience 3).

4.1. *Plate-forme expérimentale et outils*

Les résultats présentés dans cette section sont obtenus sur les équipements mis à disposition dans la plate-forme expérimentale Grid5000[CAP 06]. Il s'agit de serveurs IBM eServer 325 dotés de processeurs AMD Opteron dual-core 246 à 2GHz/1MB/400MHz et de 2GB de mémoire vive avec une interface GigaEthernet (pilote `tg3`). Ces machines physiques hébergent pour nos expériences une, deux ou quatre machines virtuelles *XEN*. Chaque machine virtuelle exécute un système d'exploitation GNU/Linux et a une partition disque dédiée (système de fichiers non-partagés). Le noyau Linux modifié pour *XEN* des domaines invités est la version `2.6.18`. Les machines physiques sont interconnectées directement via un commutateur Gigabit ethernet.

Afin de charger de manière contrôlée le processeur, nous avons développé un programme spécifique de simulation de charge utile, appellé le *softloader*. Il est constitué d'une boucle active effectuant des calculs en arithmétique entière, suivie d'une instruction de pause paramétrable (`nanosleep(long)`) qui nous permet de charger le processeur à une valeur déterminée (ex : 50%, 90%...). Pour générer et mesurer le trafic réseau nous avons utilisé l'outil de benchmark `iperf` [TIR] afin de générer des flux TCP depuis l'espace utilisateur des différentes machines virtuelles. Nous avons généralement utilisé le paramétrage par défaut de iperf. La taille de fenêtre de TCP n'a pas été modifiée directement mais via iperf. Pour superviser la charge des processeurs (ou plus exactement des *core*s) nous avons utilisé `sar`. Ainsi cet utilitaire nous permet d'obtenir les mesures de références (sans virtualisation). Nous avons utilisé une version modifiée par nos soins de `xenmon` (programmé en Python) pour mesurer cette charge sous *XEN*

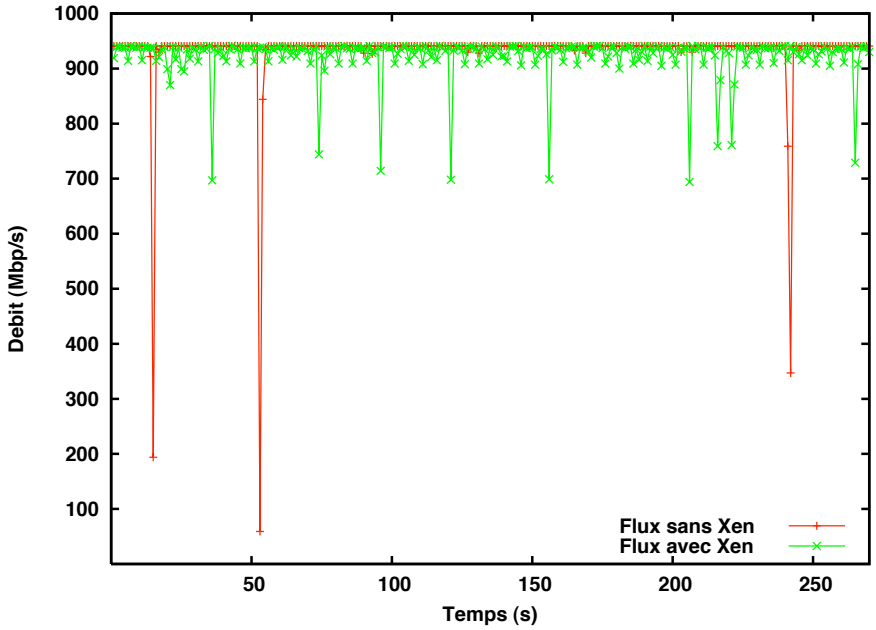

Figure 3. *Débit réseau (avec et sans XEN)*

4.2. *Résultat de l'expérience 1 : coût pour une machine virtuelle*

L'objectif de cette expérience est d'évaluer les performances de TCP dans le cas d'une machine utilisée uniquement pour la communication puis dans le cas d'une machine utilisée pour la communication et pour le calcul.

Lorsque la machine n'est sollicitée que pour communiquer, on observe que le débit réseau est très faiblement impacté par la présence de *XEN*. Il est de 938 Mbits/sec au lieu de 941 Mbits/sec maxi constaté sur cette configuration sans virtualisation). La figure 3 illustre ce modeste écart de débit, mais met aussi en évidence des chutes de débit sporadiques. La figure 4 montre le coût CPU brut de *XEN* (distance entre les courbes). On observe que le CPU n'est utilisé qu'à 30 % sans virtualisation. Le fait d'utiliser *XEN* entraîne un surcoût de 20 à 25% et les mesures mettent en évidence une instabilité de la charge (de l'ordre de 10%). Nous avons constaté un résultat comparable au niveau du serveur avec un coût CPU de base légèrement plus important (35%). Dans le cas d'une machine virtuelle exécutant simultanément une communication gigabit et un calcul saturant un cœur à 90 % nous observons que les performances réseau sont peu affectées La chute en débit est de quelques Mbps. Ces résultats encourageants sont néanmoins dus à la configuration bi-coeur des machines employées. La gestion des coeurs est faite par *XEN* et les traitements protocolaires ne sont pas systématiquement exécutés sur le même processeur.

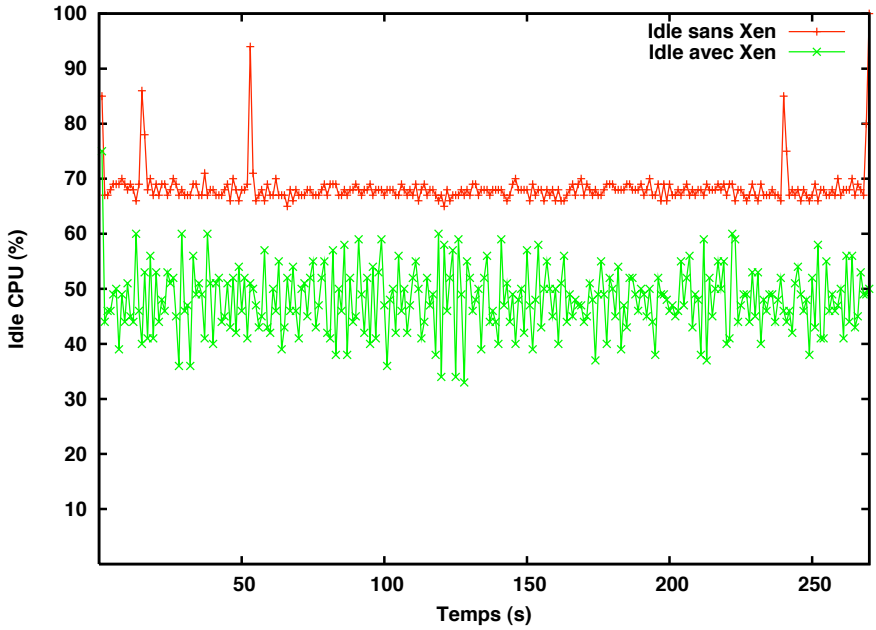

Figure 4. *Ressource CPU disponible coté client (avec et sans* XEN*).*

4.3. *Résultats de l'expérience 2 : taille des buffers*

Dans le cas de l'utilisation de liens gigabit, il est connu que le paramétrage TCP (TCP tuning) a une grande importance. Aujourd'hui les valeurs optimales sont connues et les systèmes d'exploitation activent généralement l'autotuning. Dans cette série d'expérience, la taille des fenêtres TCP a été laissée par défaut. Cette taille est automatiquement calculée et optimisée par le noyau. On note par exemple que cette taille de fenêtre n'est pas la même pour le domaine 0 (64 Ko) que pour les domaines invités ou virtuels (16 Ko) ce qui affecte les performances réseau : Communications Intra-machine
Domaine 0 vers Domaine virtuel (TCP Win 64Ko) : 610 Mbits/sec
Domaine virtuel vers Domaine 0 (TCP Win 16Ko) : 2.05 Gbits/sec

Par ailleurs, nous avons mené des expériences entre les domaine 0 et les domaines U entre 2 machines physiques distinctes.

Machine Serveur			
Machine Cliente		Domaine 0	Domaine U
	Domaine 0	941Mbits/sec	925Mbits/sec
	Domaine U	941Mbits/sec	930Mbits/sec

Le débit entre deux domaine 0 est comparable à celui que l'on pourrait obtenir avec une configuration sans *XEN*. Les débits entre les domaines 0 et les domaines virtuels sont moins bons à cause de la mauvaise sélection de la taille des fenêtres TCP. La communication entre les domaines virtuels et les domaines 0 reste très bonne et ce malgrè une taille de fenêtre TCP de 16Ko. Il faut noter que dans le cas de cette expérience la latence est très faible ($< 1ms$). La communication inter domaines virtuels (la plus commune) présente de bonnes performances avec un surcoût de 10%.

4.4. *Résultats de l'expérience 3 : Coût avec plusieurs machines virtuelles*

L'objectif de cette expérience est d'étudier le comportement dans le cas de plusieurs machines virtuelles s'exécutant sur la même machine physique. Nous avons considéré ici uniquement des configurations symétriques : N machines virtuelles clientes sur une machine physique communiquant avec N machines virtuelles clientes sur une autre machine physique serveur. Nous avons fait varier N selon les valeurs 1, 2 et 4. Nous donnons les courbes correspondant au cas de 4 machines virtuelles avec uniquement des communications (*i.e* sans charge).

	1flux 1VM		2 flux 2 VM			4 flux 4 VM				
	TCP1	Total	TCP1	TCP2	Total	TCP1	TCP2	TCP3	TCP4	Total
A vide (Mb/s)	940	940	500	410	910	350	215	155	115	835
En charge (Mb/s)	938	938	290	290	580	60	50	60	60	230

Figure 5. *Débits obtenus en fonction du nombre de machines virtuelles (en Mbits/s).*

La table 5 donne l'évolution des débits avec et sans charge dans le cas de 1, 2 et 4 machines virtuelles. On observe une réduction quasi linéaire du débit aggrégé ainsi qu'un accroissement de l'inéquité interflux dans le cas des processeurs à vide. En situation de charge, les débits s'écroulent avec l'augmentation du nombre de machines virtuelles (figures 6 et 7).

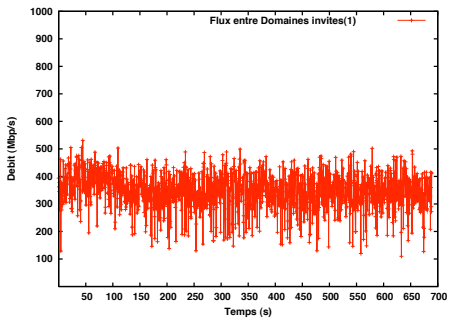

Figure 6. *Débit du flux TCP1 - cas 4 doms*

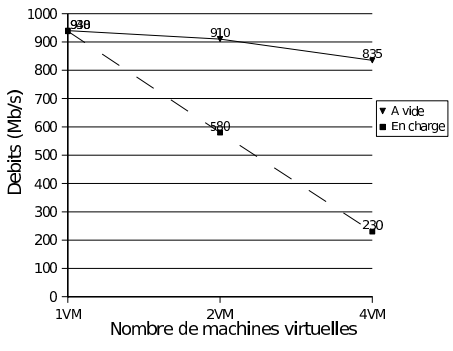

Figure 7. *Débits réseau aggrégés*

Les figures 8 et 9 explicite la répartition de la consommation des ressources dans les domaines 0 et U selon le nombre de machines virtuelles. On constate que cette consommation de ressources ne s'accroit pas linéairement. En particulier pour le domaine 0. En cas de charge, les communications semblent être fortement pénalisées au profit du calcul.

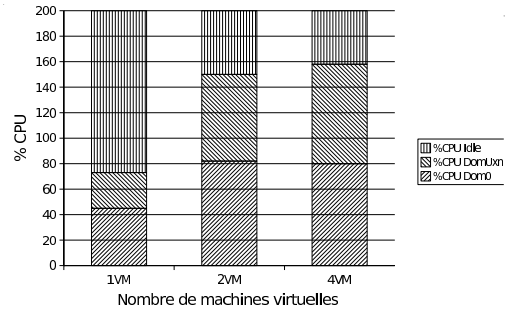

Figure 8. *Répartitions du CPU lors de transferts (à vide).*

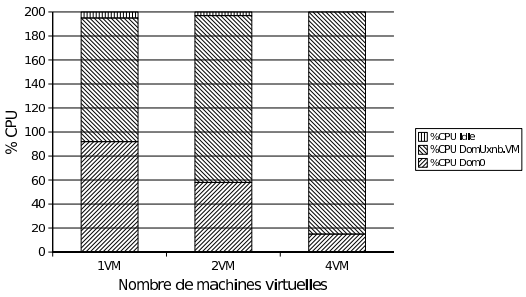

Figure 9. *Répartitions du CPU lors de transferts réseau (en charge).*

5. Travaux relatifs

La littérature dans le domaine de l'évaluation des performances réseau des systémes virtuels reste encore très restreinte. [MEN 07] montre les résultats du "profilage" de la machine virtuelle *XEN* pour déterminer où se situe les surcoûts. Ces résultats sont complémentaires aux notres même s'ils montrent des performances sous-optimales pour les connexions gigabit, étant donné que l'obtention du gigabit est dépendante de nombreux paramètres matériels. Il semble en effet que laconfiguration matérielle utilisée soit limitée. Dans [**?**] trois techniques intéressantes pour optimiser les performances réseaux de l'environnement virtuel *XEN* sont proposées. (1) les interfaces réseau virtuelles des domaines invités sont redéfinies pour profiter des fonctionnalités de haut-niveau disponibles sur les cartes réseau moderne (*offloading*). (2) l'implémentation du chemin de transfert des données entre le pilote du domaine invité et celui du Dom 0 sont optimisées. (3) le domaine invité a la possibilité d'accéder directement à la mémoire virtuelle (via des *superpages* ou des pages globales mappées).

6. Conclusions et travaux futurs

Les résultats obtenus montrent que le débit d'une connexion seule sur un lien Gigabit n'est quasiment pas affecté par la virtualisation même dans le cas d'une machine chargée à 90% (sur un cœur). Néanmoins, ce débit est obtenu au prix d'une consommation CPU supérieure et d'un ajustement des tailles de buffers. Lorsque l'on multiplie le nombre de machines virtuelles nos mesures montrent que le surcoût dû à la

virtualisation augmente linéairement avec le nombre de machines. On observe également un accroissement important de l'inéquité entre flux ce qui risque de poser des problèmes dans le cas de l'utilisation de la virtualisation dans les routeurs.

En termes de travaux futurs, nous nous proposons de mener une étude détaillée conjointe de l'ordonnancement des flux au niveau du domaine 0 et de l'ordonnancement des tâches dans *XEN*. Nous souhaitons aussi étudier les coûts lors de l'utilisation d'un routeur en domaine 0. Par ailleurs, les machines que nous avons utilisé pour nos expérimentations ne proposent pas le support de la virtualisation matérielle. Il serait interressant de refaire les mêmes mesures sur des machines offrant le support matériel de la virtualisation. Notre prochain objectif est d'évaluer le surcoût de la virtualisation sur le délai et la jigue. Enfin, à plus long terme nous souhaitons tirer avantage des cartes d'interfaces réseau programmables (équipés de *Network Processor*) en y déportant le switch/router virtuel ce qui devrait soulager le ou les processeurs physiques pour l'arbitrage des flux issus de domaines invités multiples.

7. Bibliographie

[BAR 03] BARHAM P., DRAGOVIC B., FRASER K., HAND S., HARRIS T., HO A., NEUGEBAUER R., PRATT I., WARFIELD A., « Xen and the art of virtualization », *SOSP '03 : Proceedings of the nineteenth ACM symposium on Operating systems principles*, ACM Press, 2003, p. 164–177.

[CAP 06] CAPPELLO F., DESPREZ F., DAYDE M., JEANNOT E., JEGOU Y., LANTERI S., MELAB N., NAMYST R., VICAT-BLANC PRIMET P., RICHARD O., CARON E., LEDUC J., MORNET G., « Grid5000 : a nation wide experimental grid testbed », *in International Journal on High Performance Computing Applications*, , 2006.

[GEL 03] GELINAS J., « Virtual Private Servers and Security Contexts », *http ://www.solucorp.qc.ca/miscprj/s_context.hc*, 2003.

[HOX 02] HOXER H.-J., BUCHACKER K., SIEH V., « Implementing a User Mode Linux with Minimal Changes from Original Kernel », *Proceedings of 9th International Linux System Technology Conference*, Cologne, Germany, September 2002.

[LES 96] LESLIE I. M., MCAULEY D., BLACK R., ROSCOE T., BARHAM P., D. EVERS R., FAIRBAIRNS, HYDEN E., « The design and implementation of an operating system to support distributed multimedia applications », *IEEE Journal on Selected Areas In Communications, 14(7) :12801297*, , 1996.

[MEN 07] MENON A., ZWAENEPOEL W., SANTOS J. R., TURNER Y., JANAKIRAMAN J., « Diagnosing Performance overheads in the Xen virtual machine environment », *Xen Summit 4, Yoktown, NY, USA*, , 2007.

[PRA 05] PRATT I., FRASER K., « Xen 3.0 and the Art of Virtualization », *Linux Symposium*, , 2005.

[QUM] QUMRANET, « KVM », `http://kvm.qumranet.com/kvmwiki`.

[TIR] TIRUMALA A., QIN F., DUGAN J., FERGUSON J., GIBBS K., « iperf : testing the limits of your network », `http://dast.nlanr.net/Projects/Iperf/`.