Migration dynamique d'applications réparties virtualisées dans les fédérations d'infrastructures distribuées

Djawida Dib

IFSIC
Université de Rennes 1
Campus de Beaulieu
35042 RENNES CEDEX

IRISA – INRIA
Équipe MYRIADS
Campus de Beaulieu
35042 RENNES CEDEX

Migration dynamique d'applications réparties virtualisées dans les fédérations d'infrastructures distribuées

Rapport de Stage de Master Recherche

Djawida DIB
Djawida.Dib@irisa.fr

Supervisé par **Pierre RITEAU** et **Christine MORIN**
Équipe MYRIADS, IRISA, INRIA Rennes - Bretagne Atlantique
{Pierre.Riteau,Christine.Morin}@irisa.fr

Juin 2010

Table des matières

Introduction

Le cloud computing est un nouveau concept dont les prémices remontent à la technologie des grilles de calcul, utilisées pour le calcul scientifique. Une grille de calcul est une fédération de ressources informatiques hétérogènes (postes de travail, grappes de serveurs, ...), distribuées géographiquement et appartenant à différentes organisations. Les grilles informatiques permettent le calcul distribué à large échelle en exploitant un grand nombre de ressources. La principale différence entre une grille de calcul et un cloud est la manière dont les ressources sont disponibles pour l'utilisateur.

Le cloud computing fait ainsi référence à l'utilisation des capacités de calcul d'ordinateurs distants, où l'utilisateur dispose d'une puissance informatique considérable sans avoir à posséder des unités puissantes. L'utilisateur d'un cloud a l'impression d'avoir en permanence accès à des ressources de calcul et/ou de stockage illimitées. Il peut donc adapter le niveau des ressources suivant la charge de ses applications. Les services qu'offre un cloud sont facturés proportionnellement à l'usage des infrastructures physiques (capacité des ressources, taille des données échangées avec d'autres infrastructures) et à la période d'utilisation.

L'approche du cloud computing s'appuie sur la technologie de la virtualisation, où la virtualisation est un ensemble de techniques permettant de faire fonctionner plusieurs systèmes, isolés les uns des autres, sur un seul système physique. Grâce à la virtualisation un cloud peut adapter son environnement informatique à l'évolution de ses activités. Un service offert par un cloud est constitué d'un ensemble de machines virtuelles, utilisant la même infrastructure physique, qui s'adaptent à la charge applicative nécessaire pour l'utilisateur.

La virtualisation offre une fonctionnalité nommée *migration à chaud* qui permet de migrer des machines virtuelles d'un nœud physique à un autre. La migration à chaud d'une machine virtuelle peut se faire sans interruption de service, ce qui rend le processus totalement transparent. La migration à chaud peut être utilisée pour mettre en œuvre l'équilibrage de charge, afin d'optimiser l'utilisation des ressources et de mieux gérer et économiser l'énergie. Elle est susceptible d'être très utile pour la maintenance d'infrastructures physiques. Elle permet également d'améliorer la tolérance aux fautes : si une défaillance imminente est suspectée, le problème peut être résolu avant d'interrompre le service.

Dans le cadre des clouds, les prix d'hébergement de machines virtuelles peuvent varier au fil du temps et d'un cloud à un autre, ce qui rend la migration des machines virtuelles entre clouds un véritable atout économique du point de vue de l'utilisateur.

Cependant les limitations de la migration à chaud s'inscrivent dans le cas où nous souhaitons déplacer une machine virtuelle d'une infrastructure à une autre, ce que nous appelons dans la suite de ce document la *migration inter-site*. Dans ces circonstances la machine virtuelle déplacée perd ses anciennes communications à cause de conflits d'adressage ou de problèmes de routage.

Ce rapport traite de la problématique de migration à chaud inter-site. Plus précisément il traite

de la prise en compte des schémas de communication de machines virtuelles dans la migration afin d'optimiser leur placement dans les différentes infrastructures disponibles.

Ce rapport est composé de quatre parties. Dans le premier chapitre, nous faisons un état de l'art des techniques permettant de réaliser la migration à chaud inter-site de machines virtuelles. Le deuxième chapitre explique en détail l'objectif du travail de recherche de ce stage, qui a trait à la migration dynamique d'applications réparties virtualisées dans les fédérations d'infrastructures distribuées telles que les clouds et les grilles de calcul. Dans le troisième chapitre, nous présentons l'architecture des réseaux virtuels étudiés, à savoir *VNET* et *IPOP*, ainsi que les évaluations réalisées sur les mises en œuvres existantes.

Le quatrième chapitre aborde notre contribution aux systèmes de gestion dynamique d'applications distribuées. Notre contribution traite plus précisément les techniques de migration de machines virtuelles de manière autonome, en se basant sur les schémas de communication, afin d'utiliser au mieux les ressources disponibles et de réduire le coût ou le temps d'exécution de l'application. Nous montrons par la suite les aspects les plus techniques et l'évaluation de notre mise en œuvre. Enfin, nous concluons ce rapport puis donnons quelques perspectives de recherche pour l'extension de ces travaux.

Chapitre 1

État de l'art

Ce premier chapitre présente les mécanismes actuellement disponibles de migration à chaud de machines virtuelles entre différentes infrastructures de type cloud computing. Dans le paragraphe 1.1 nous commençons par présenter le concept de virtualisation et expliquons le processus de migration à chaud des machines virtuelles. Nous décrivons dans le paragraphe 1.2 les problèmes rencontrés lors de la migration inter-site des machines virtuelles. Dans le paragraphe 1.3 nous détaillons les différentes approches permettant de mettre en oeuvre un système de migration de machines virtuelles entre différentes infrastructures de type cloud computing. Nous définissons dans le paragraphe 1.4 la notion de prise en compte des schémas de communication pour la migration des machines virtuelles exécutant une application distribuée. Enfin nous concluons ce chapitre en dressant un bilan autour des éléments importants de cette étude.

1.1 Virtualisation et migration à chaud

1.1.1 Virtualisation

La virtualisation est une technologie permettant l'exécution de plusieurs systèmes d'exploitation et/ou plusieurs applications sur le même ordinateur, ce qui accroît l'utilisation et la flexibilité du matériel. Un même ordinateur peut ainsi permettre l'exécution de plusieurs machines virtuelles contenant différents systèmes d'exploitation et/ou différentes applications isolées les unes des autres.

La mise en œuvre d'une machine virtuelle *(virtual machine* ou *VM)* nécessite l'ajout d'une couche logicielle à la machine physique. L'emplacement de cette couche dépend du type d'architecture de la machine virtuelle [1]. Il existe deux types de technologies de machine virtuelle : les machines virtuelles applicatives *(process virtual machines)* et les machines virtuelles système *(system virtual machines)*.

Dans une machine virtuelle applicative, la nouvelle couche logicielle s'appelle *runtime* (abréviation de *runtime software*), et se place entre le système d'exploitation et les applications utilisateur. La machine virtuelle applicative est un programme qui émule dans un système d'exploitation un environnement standardisé, isolé et sécurisé. Cela permet au concepteur de l'application de ne pas avoir à rédiger plusieurs versions de son logiciel pour qu'il puisse l'exécuter dans tous les environnements. La création et la terminaison de ce type de machine virtuelle se fait lors de la création et la terminaison du processus exécutant l'application virtualisée. L'exemple le plus connu de machine virtuelle applicative est la machine virtuelle Java, *JVM* [2].

En revanche, dans une machine virtuelle système la nouvelle couche logicielle se place entre le matériel et le système d'exploitation et s'appelle *hyperviseur* ou *moniteur de machine virtuelle*, en anglais *hypervisor* ou *virtual machine monitor (VMM)*. Une machine virtuelle système fournit un environnement complet qui a son propre système d'exploitation (cf. figure 1.1).

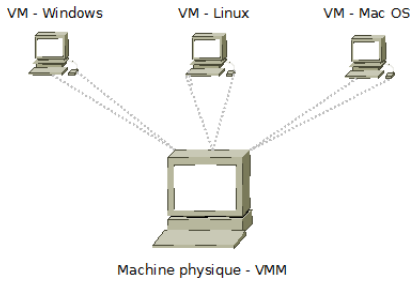


FIGURE 1.1 – Concept de virtualisation niveau système

Le système d'exploitation qui s'exécute dans une machine virtuelle système est appelé *invité*, en anglais *guest*, tandis que la plateforme sous-jacente qui supporte la machine virtuelle est l'*hôte*, en anglais *host*. Une machine virtuelle système offre une copie du matériel et se comporte exactement comme un ordinateur physique contenant ses propres processeur, mémoire, disque dur et cartes d'extension (carte d'interface réseau). En conséquence, le système d'exploitation invité est généralement incapable de faire la différence entre une machine virtuelle et une machine physique.

Dans ce qui suit, nous nous intéressons principalement aux machines virtuelles système, car celles ci offrent certains avantages par rapport aux machines virtuelles applicatives tels que :

- Un support avancé de la migration.
- Un support de l'exécution d'applications patrimoniales *(legacy application)*[1], ainsi un environnement d'exécution ne requiert pas de modification pour être exécuté sur une machine virtuelle système.

Il existe deux types d'architecture de machine virtuelle système [3]. Dans le type I le VMM se trouve directement au-dessus de la couche matérielle. Au-dessus du VMM se trouvent les différentes VMs. Sur une de ces VMs s'exécute le système d'exploitation *(Operating System* ou *OS)* de la machine hôte qui gère le partage des ressources entre les autres VMs. Les autres VMs exécutent des OS moins privilégiés. Cette architecture est utilisée notamment par Xen [4].

Dans le type II, c'est l'OS de la machine hôte qui se trouve directement sur la couche matérielle. Au-dessus de celui-ci le VMM prend place. Au-dessus du VMM se trouve les OS invités des différentes VMs. Cette architecture est utilisée par des hyperviseurs tel que VMware Workstation [5], KVM [6] ou VirtualBox [7].

1.1.2 Migration à chaud

La migration à chaud permet de déplacer une machine virtuelle d'un nœud physique à un autre sans interruption de service (cf. figure 1.2), ce qui rend le processus totalement transparent pour l'utilisateur.

1. Les applications patrimoniales sont des applications qui ne sont pas modifiables, généralement parce qu'elles sont vieilles et que les ressources nécessaires pour les modifier sont trop importantes.

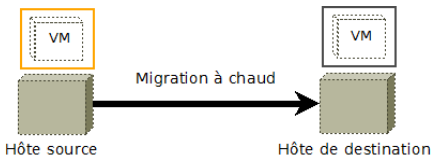


FIGURE 1.2 – Principe de la migration

Le processus de migration à chaud d'une machine virtuelle d'un hôte source vers un hôte de destination se compose de six étapes principales [8, 9] :

1. **Réservation**
   L'hôte source envoie la requête de migration à l'hôte de destination et attend sa confirmation de disponibilité des ressources. L'hôte de destination réserve par la suite la mémoire nécessaire à la machine virtuelle.
2. **Copie itérative de pages mémoire**
   Pendant la première itération toutes les pages mémoire de la machine virtuelle source sont copiées sur l'hôte de destination dans la mémoire qui lui est allouée. Dans les itérations ultérieures, seulement les pages mémoire modifiées *(dirty pages)* durant la phase de transfert précédente sont copiées.
3. **Stop et copie**
   La machine virtuelle est mise en pause afin de permettre une ultime copie des pages mémoire modifiées. L'hôte transfère également les registres processeur ainsi que l'état des périphériques à la machine de destination. À la fin du transfert, la VM cible devient une copie parfaite de la VM source.
4. **Redirection**
   L'hôte de destination indique à l'hôte source qu'il a correctement reçu l'image de la VM.
5. **Activation de la machine virtuelle**
   La machine virtuelle migrée est activée et peut continuer son exécution.
6. **Mise à jour des informations réseau**
   Un message de mise à jour est envoyé au switch physique afin d'adapter sa table de routage. De ce fait, les paquets à destination de la machine virtuelle ne passent plus par le même port physique.

1.2 Problèmes réseau lors de la migration

Dans cette section, nous définissons les problèmes liés à la migration inter-site des machines virtuelles appartenant à la même grappe virtuelle. Une grappe virtuelle est un ensemble de machines virtuelles utilisées pour l'exécution d'une application distribuée. Initialement nous supposons que toutes les VMs de la grappe virtuelle se trouvent sur le même cloud (cf. figure 1.3).

Ensuite, nous considérons un scénario où nous désirons migrer certaines VMs sur un autre cloud. Ceci nous conduit à trois cas problématiques possibles. Le premier cas se présente si la migration se fait seulement sur une partie des VMs de la grappe virtuelle (cf. figure 1.4). Dans ce cas, il y a deux possibilités :

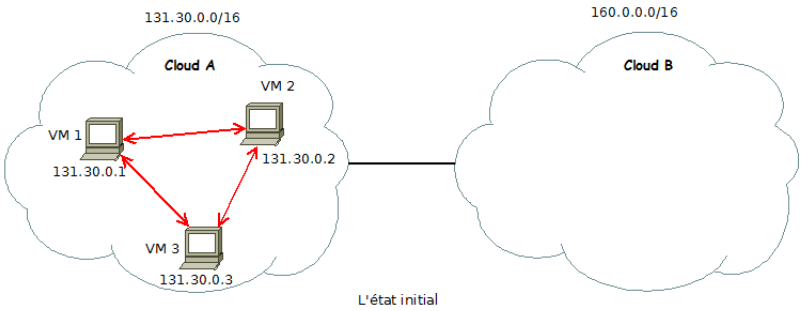


FIGURE 1.3 – L'état initial de la grappe virtuelle

1. La première possibilité est de garder les adresses IP des VMs migrées. Dans ce cas l'adresse IP de la VM migrée est la même mais le chemin à suivre pour lui transmettre des messages n'est plus le même. Il peut y avoir également un conflit d'adressage avec d'autres VMs déjà hébergées sur le cloud de destination (une de ces VMs peut avoir la même adresse que celle de la VM migrée).
2. La seconde possibilité consiste à mettre à jour les adresses IP des VMs migrées. Dans ce cas l'ensemble des VMs est incapable de connaître la nouvelle adresse IP de la VM migrée.

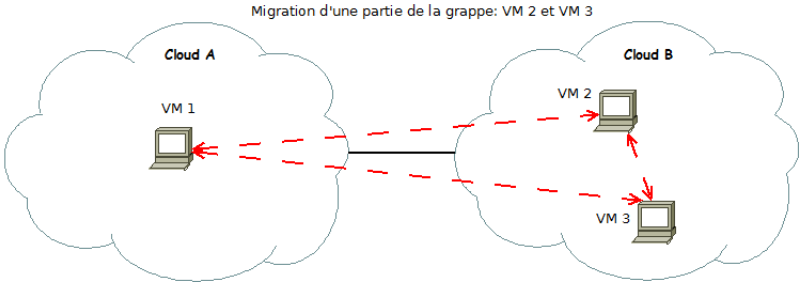


FIGURE 1.4 – Migration partielle de la grappe virtuelle

Le deuxième cas problématique consiste à migrer entièrement la grappe virtuelle (cf. figure 1.5). Ceci peut être considéré comme un cas particulier du premier. Il est résoluble si les trois conditions, ci-dessous, seront satisfaites :

1. Éliminer tous les conflits entre les adresses IP des VMs migrées et celles des VMs déjà hébergées sur le cloud de destination.
2. Éliminer également tous les conflits entre les adresses Ethernet.
3. Ne pas utiliser des routeurs entre les VMs à migrer.

Le dernier cas problématique apparaît lors de la migration d'une VM communiquant avec un utilisateur extérieur ne faisant pas partie de la grappe virtuelle (cf. figure 1.6). Ceci entraîne deux possibilités similaires à celles vues dans le premier cas problématique. Si la VM garde son adresse IP, l'utilisateur envoie les messages à destination de la VM au cloud source au lieu de les envoyer au cloud de destination. Par contre si la VM met à jour son adresse IP, l'utilisateur est incapable de la connaître.

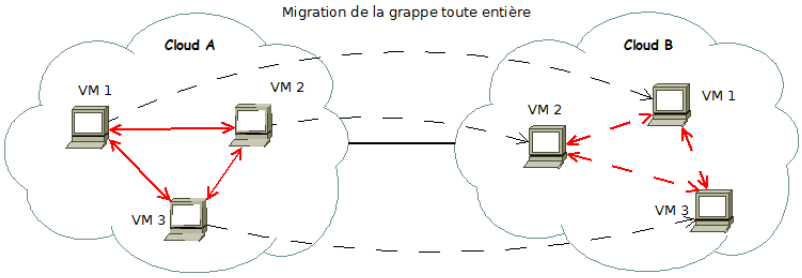


FIGURE 1.5 – Migration entière de la grappe virtuelle

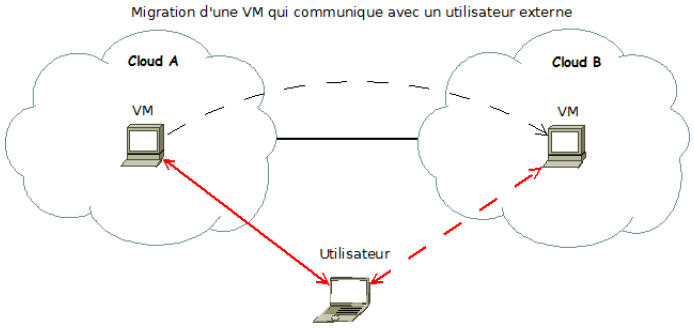


FIGURE 1.6 – Migration d'une machine virtuelle communiquant avec un utilisateur extérieur.

Dans ces trois cas problématiques la communication entre les machines virtuelles s'interrompt après la migration, puisque les adresses IP des VMs ne sont plus valides à cause des conflits d'adressage ou des problèmes de routage. Non seulement la recherche d'un moyen pour résoudre ces problèmes est nécessaire, mais il faut aussi s'ingénier à optimiser les performances. Par exemple, si les machines VM 2 et VM 3 de la figure 1.4 communiquent, elles doivent utiliser l'infrastructure physique du cloud B sans avoir à passer par le cloud A.

Dans ce qui suit nous détaillerons les différents mécanismes permettant la migration inter-site de machines virtuelles, et présenterons leurs caractéristiques.

1.3 Mécanismes permettant la migration inter-site

1.3.1 Protocole Mobile IPv6

Le but du protocole Mobile IPv6 [10] est de rendre joignable à tout instant un périphérique mobile. Le même principe peut s'appliquer par analogie pour rendre joignable à tout instant une machine virtuelle migrable [11, 12].

La figure 1.7 présente le principe de fonctionnement du protocole Mobile IPv6. Initialement le nœud mobile est lié à un réseau géré par un agent mère [2]. Après le déplacement du nœud mobile, il sera attaché à un nouveau réseau administré par un agent étranger [3]. Dans un premier temps le nœud mobile envoie un message de mise à jour à son agent mère pour l'informer de son nouvel

2. Un agent mère est un routeur situé dans le réseau administratif du nœud mobile.
3. Un agent étranger est un routeur situé dans le réseau visité par le nœud mobile

emplacement. Si l'agent mère détecte un nouveau correspondant souhaitant communiquer avec le nœud mobile, il transmet d'abord le paquet à ce dernier et envoie à son tour un message de mise à jour au nœud correspondant. Le nœud correspondant crée par la suite un tunnel direct entre lui et le nœud mobile, évitant ainsi de passer par l'agent mère.

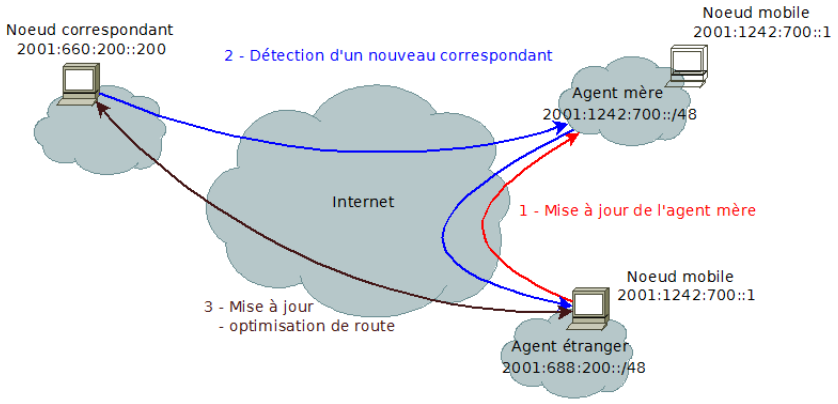


FIGURE 1.7 – Principe du protocole mobile IPv6

1.3.2 Réseaux virtuels

Les réseaux virtuels prennent de plus en plus place dans la communauté réseau et offrent de nouvelles perspectives pour l'internet du futur, tout en permettant le déploiement de différentes architectures et de différents protocoles sur une infrastructure physique partagée [13].

La virtualisation réseau consiste à virtualiser à la fois les noeuds et les liens. La virtualisation des nœuds est fondée sur l'isolation et le partitionnement des ressources du nœud physique. D'autre part, chaque lien physique peut transporter plusieurs liens virtuels. L'interconnexion de ces noeuds virtuels et liens virtuels permet la création d'un réseau virtuel. Ainsi les réseaux virtuels permettent d'optimiser l'utilisation des infrastructures physiques et de mieux les gérer. Ce concept peut également avoir des intérêts économiques. Par exemple le partage d'une infrastructure physique entre différents opérateurs réseaux réduit considérablement le coût de propriété.

L'architecture de la plupart des réseaux virtuels s'appuie sur un réseau *overlay* [14]. Un réseau overlay est un réseau au-dessus d'un réseau physique ou d'un autre réseau overlay. Les réseaux overlay sont généralement décentralisés, distribués et sans organisation hiérarchique. Dans ce cas ils sont appelés des réseaux pair-à-pair ou P2P (*peer-to-peer*). Chaque nœud d'un réseau overlay, appelés *pair*, joue à la fois le rôle de serveur et de client. L'organisation dans un tel réseau repose sur l'ensemble des pairs, sans avoir une entité chargée de l'administrer. La figure 1.8 schématise les trois couches réseau : réseau physique, réseau overlay et réseau virtuel.

Un réseau virtuel utilise les services offerts par le réseau overlay sous-jacent telle qu'une table de hachage distribuée (ou DHT pour *Distributed Hash Table*) [4].

La figure 1.9 illustre le principe de fonctionnement d'un réseau virtuel. Nous considérons deux machines virtuelles VM1 et VM2, appartenant au même réseau virtuel, mais hébergées sur deux hôtes physiques qui appartiennent à des réseaux physiques différents. Quand VM1 envoie un paquet à VM2, l'hôte physique qui l'héberge (hôte 1) l'intercepte.

4. Une DHT est une base de données distribuée où les données sont stockées dans le format (clé, valeur).

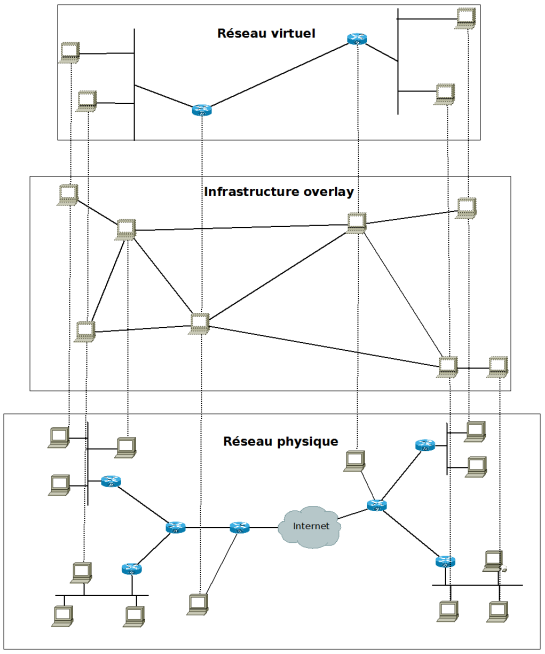


FIGURE 1.8 – Schéma des couches réseau permettant le fonctionnement d'un réseau virtuel.

L'hôte 1 cherche l'adresse IP de l'hôte hébergeant une VM dont l'adresse est égale à l'adresse de destination du paquet intercepté en utilisant la DHT de l'overlay. Avant d'envoyer le paquet sur le réseau physique, l'hôte 1 l'encapsule dans un nouveau paquet qui a pour adresse source son adresse, pour adresse de destination l'adresse de l'hôte 2 (l'hôte qui héberge la VM2) et considère le paquet original comme données. L'hôte 2 désencapsule le paquet reçu et l'injecte dans le réseau virtuel. La VM2 reçoit ainsi le paquet envoyé par VM1 d'une façon totalement transparente.

Les réseaux virtuels peuvent par leurs fléxibilité résoudre les problèmes liés à la migration inter-site cités dans la section 1.2, pour cela il doivent mettre à jour les informations correspondantes dans la DHT.

Plusieurs travaux ont été menés sur l'architecture des réseaux virtuels. Nous récapitulons les principales caractéristiques des différentes architecture proposées dans le tableau 1.1. Certaines architectures sont de couche 2, ce qui veut dire que les communications se font par exemple à travers des paquets Ethernet. D'autres sont plutôt de couche 3, les communications se font évidemment à travers des paquets IP. Nous distinguons également deux types de réseaux virtuels, les réseaux virtuels stables et les réseaux virtuels reconfigurables.

Un réseau virtuel stable utilise les mêmes liens et les mêmes noeuds pendant toute sa durée de vie, donc sa topologie ne change pas. L'administrateur d'un tel réseau doit réserver toutes les ressources nécessaires pour effectuer ses tâches à la configuration initiale. Les réseaux virtuels stables sont peu adaptés à la migration dynamique entre clouds car ils nécessitent de prévoir l'architecture complète du réseau dès sa création.

En revanche, la topologie d'un réseau virtuel reconfigurable peut changer, en fonction des ressources disponibles sur les infrastructures physiques utilisées, afin d'améliorer les performances ou de réduire le coût d'utilisation. La plupart des réseaux virtuels étudiés sont reconfigurables. Ces derniers suscitent beaucoup d'intêrets car ils permettent de résoudre le problème de perte de com-

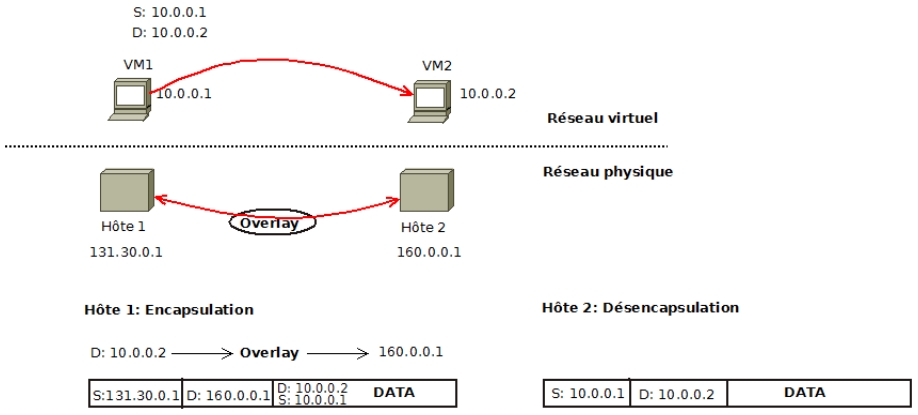


FIGURE 1.9 – Principe de fonctionnement d'un réseau virtuel

	Configuration	Niveau d'adressage	Migration
Violin [15]	Reconfigurable	Couche 3	Migration à la demande
ViNe [16]	Reconfigurable	Couche 3	Migration non mentionnée
SoftUDC VNET [17]	Reconfigurable	Couche 2	Migration possible mais pas discutée pour les réseaux étendus.
Virtuoso VNET [18]	Reconfigurable	Couche 2	Migration possible
IPOP [19]	Reconfigurable	Couche 3	Migration possible
PVC [20]	Stable	Couche 2	Migration non discutée
N2N [21]	Reconfigurable	Couche 2	Migration non discutée
OCALA [22]	Non mentionné	Couche 3	Migration non discutée

TABLE 1.1 – Comparaison des caractéristiques de réseaux virtuels étudiés

munication lors d'une migration inter-site.

1.3.3 Bilan des mécanismes étudiés

Nous avons dressé un bilan des mécanismes étudiés qui permettent la migration inter-site des machines virtuelles dans le tableau 1.2. Un réseau virtuel est plus complexe à mettre en œuvre que le mécanisme utilisant le protocole Mobile IPv6, néanmoins il est indépendant de l'infrastructure physique et ne nécessite aucun équipement supplémentaire. À l'inverse, le protocole Mobile IPv6 exige que tous les équipements supportent ce protocole et nécessite un agent dans chaque réseau.

1.4 Prise en compte des schémas de communication pour la migration d'applications distribuées

Une application distribuée est une application composée de plusieurs processus communiquant entre eux et pouvant être distants géographiquement. Les composants d'une application distribuée

	Mobile IPv6	Réseau virtuel
Mise en œuvre	Simple	Complexe
Équipement	Agent mère	Aucun
Protocole	Mobile IPv6	Spécifique à chaque réseau virtuel

TABLE 1.2 – Bilan des mécanismes étudiés

peuvent communiquer entre eux à travers le passage des messages, en utilisant par exemple MPI[5], ou à travers d'autres mécanismes.

Les processus d'une application distribuée ne communiquent pas forcément entre eux de la même façon. Par conséquent les applications distribuées peuvent avoir des schémas de communication différents. Pour assurer de bonnes performances, la migration inter-site de telles applications doit tenir compte de ces schémas de communication. En effet, migrer sur deux clouds différents des processus qui communiquent beaucoup entre eux pénaliserait les performances. Par exemple dans l'application CG des NAS Parallel Benchmarks [23], les nœuds forment des groupes de communication. Dans ce cas, il faudrait faire en sorte de migrer uniquement des groupes de communication complets.

Plusieurs travaux ont été réalisés pour la détection de schémas de communication des applications distribuées [24, 25, 26, 27, 28]. La plupart d'entre eux se basent sur la librairie MPI. Les mécanismes utilisés en [24] et [25] consistent à apporter des modifications à cette librairie. Dans les travaux réalisés en [26] et [27] la détection de schémas de communication ne nécessite aucune modification de la librairie MPI, mais elle se fait en utilisant les traces de communication MPI. Enfin, les travaux menés dans [28] sont indépendant de la librairie MPI. Ils se fondent sur l'analyse du trafic réseau en capturant les paquets TCP. Le mécanisme de capture utilisé est détaillé dans [29].

Pour réaliser une migration dynamique tenant compte de ces schémas de communication d'une façon transparente, il serait plus intéressant de détecter les schémas de communication en analysant le trafic réseau des machines virtuelles depuis l'hyperviseur. Ces informations sur le trafic nous permettraient de détecter les groupes de communication éventuels formés par l'application distribuée.

1.5 Conclusion

Dans ce chapitre, nous avons montré que la migration à chaud, permise par l'utilisation de technologies de virtualisation dans les infrastructures de type cloud computing, présente un intérêt important. Nous nous sommes intéressés aux avantages présentés par la migration à chaud de machines virtuelles entre clouds. Nous avons indiqué les problèmes importants qui se posent dans le contexte de la migration à chaud des machines virtuelles entre réseaux physiques distincts. Ces problèmes provoquent la perte des communications en cours à cause de conflits d'adressage ou de problèmes de routage.

Nous avons étudié deux mécanismes permettant de résoudre ces problèmes réseaux. Le premier s'appuie sur l'utilisation du protocole Mobile IPv6 et le second se base sur la création de réseaux virtuels, en se fondant sur l'utilisation des réseaux overlay. Enfin, nous avons introduit la présence de schémas de communication entre processus d'une application distribuée. Afin de garantir un

5. MPI est une librairie d'échanges de messages (Message Passing Interface) pour machines parallèles.

bon niveau de performances, ces schémas de communication doivent être pris en compte lors de la migration à chaud de machines virtuelles.

Dans le chapitre suivant, nous définissons les différents objectifs de ce stage. Le travail s'oriente autour de la conception d'un système de migration dynamique de machines virtuelles respectant les schémas de communications.

Chapitre 2

Objectifs de stage

Les travaux de recherche effectués dans le cadre de ce stage ont trait à la gestion dynamique d'applications distribuées virtualisées dans les fédérations d'infrastructures distribuées telles que les clouds et les grilles de calcul. Plus précisément, ils sont axés autour de la prise en compte des schémas de communications dans la migration dynamique de machines virtuelles.

À l'instar du bilan que nous avons dressé dans le paragraphe 1.3.3, nous nous intéressons aux mises en œuvre de réseaux virtuels vu qu'elles ne requièrent aucun équipement supplémentaire.

L'objectif central de ces travaux de recherche est de concevoir un système de migration dynamique de machines virtuelles respectant les schémas de communication et la disponibilité des ressources dans différents environnements, afin d'améliorer les performances ou de réduire le coût d'exécution d'une application.

La première partie consiste à effectuer des expérimentations sur les mises en œuvre de réseau virtuel existantes afin de les évaluer et d'étudier leur support de la migration à chaud de machines virtuelles. Nous avons choisi les deux réseaux virtuels VNET et IPOP, parce qu'ils sont tous les deux reconfigurables et disponibles en ligne.

La deuxième partie s'attache à proposer un modèle permettant la migration dynamique de machines virtuelles. Nous commençons d'abord par mettre en œuvre un système de détection des schémas de communication, qui sera considéré comme module de notre modèle de migration dynamique. Ce système doit agir de façon totalement transparente en analysant le trafic réseau de façon passive. À la différence du mécanisme utilisé en [28] qui ne capture que les paquets TCP, notre système s'intersse au trafic IP en général, et est ainsi capable d'analyser les communications d'applications fondées sur d'autres protocoles de transport tels que UDP.

La validation de notre proposition sera réalisée sur Grid'5000 [30]. Grid'5000 est une grille expérimentale répartie sur 9 sites en France, pour la recherche dans les systèmes parallèles à large-échelle et les systèmes distribués.

Notre évaluation porte sur la fiabilité des résultats obtenus, en les comparant avec les résultats des travaux précédents, et sur l'impact de l'exécution de notre système sur les performances en temps d'exécution.

Le chapitre suivant présente l'architecture des réseaux virtuels IPOP et VNET ainsi que l'évaluation réalisée sur leurs mises en œuvre disponibles.

Chapitre 3

Les Réseaux virtuels VNET et IPOP

Dans ce chapitre, nous expliquons en détail l'architecture des réseaux virtuels VNET et IPOP qui nous servent à tester la mise en œuvre de notre modèle de migration dynamique de machines virtuelles. Nous décrivons également l'ensemble des expérimentations effectuées sur chacun. Les réseaux virtuels VNET et IPOP sont de couche 2 et de couche 3 respectivement, ce qui enrichi nos expérimentations.

3.1 VNET

3.1.1 Architecture

VNET [31, 18] est un réseau virtuel de couche 2 faisant illusion à un utilisateur exploitant une machine virtuelle distante que la machine est dans son propre réseau local *(Local Area Network ou LAN)*. Étant donné que ce réseau virtuel est de couche 2, la machine virtuelle peut être migrée d'un site à un autre en gardant la même adresse IP.

Le mécanisme spécifique utilisé est l'extraction et l'injection des paquets Ethernet transmis par la carte réseau virtuelle. Chaque machine physique (hôte) susceptible d'instancier une machine virtuelle exécute un seul *serveur VNET*[1]. Une machine dans le réseau local de l'utilisateur exécute aussi un serveur VNET. Cette machine est vue en tant que proxy.

La figure 3.1 montre la configuration de démarrage typique de VNET pour quatre hôtes, où chacun peut supporter plusieurs VMs. Chaque serveur VNET s'exécutant sur un hôte étranger[2] est connecté, en utilisant une connexion TCP, avec le serveur VNET qui s'exécute sur le proxy. Les serveurs VNET tournant sur les hôtes et le proxy ouvrent leurs interfaces virtuelles en mode promiscuité[3]. Chaque serveur VNET a une table de routage, lui permettant de déterminer où envoyer les paquets capturés de l'interface ou reçus depuis la connexion TCP.

Cette topologie en étoile est seulement la configuration initiale afin de fournir une connectivité pour les VMs. Des liens et règles de routage supplémentaires peuvent être ajoutés ou supprimés à n'importe quel moment, pour rendre l'adaptation de topologie possible.

La figure 3.2 illustre une configuration de VNET dynamiquement adaptée au modèle de communication. Ce modèle représente un échange entre voisins dans une application de topologie en

1. Un serveur VNET est un script pour créer les hôtes et les serveurs virtuels nécessaires [32].
2. Un hôte étranger est un hôte appartenant à un réseau physique différent de celui de l'utilisateur.
3. Le mode promiscuité se réfère à une configuration de la carte réseau, qui permet à celle-ci d'accepter tous les paquets qu'elle reçoit, même si ceux-ci ne lui sont pas adressés.

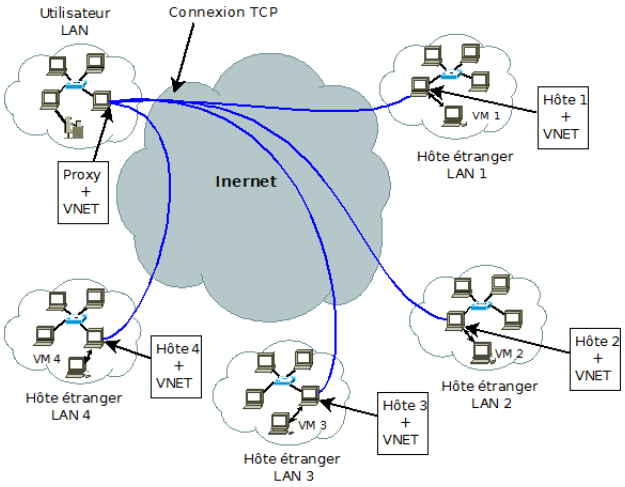


FIGURE 3.1 – Topologie de démarrage de VNET

anneau pour quatre VMs. L'application commence l'exécution en topologie étoile (pointillés) centrée sur le proxy. En continuant l'exécution, VTTIF [4] infère qu'une topologie en anneau existe. VNET ajoute donc quatre liens pour former la topologie en anneau entre les quatre serveurs VNET.

Les liens ajoutés forment *une topologie en voie rapide*, vu qu'ils permettent une communication plus rapide entre les composants de l'application. Par exemple, VM1 peut ainsi communiquer directement avec VM2 sur une connexion TCP, au lieu de rediriger le trafic correspondant sur le proxy. Cependant, la topologie en étoile résiliente doit être maintenu en permanence. La topologie en voie rapide et ses règles de routage sont modifiés suivant les besoins afin d'améliorer les performances.

3.1.2 Évaluation

Dans cette section, nous commençons par expliquer quelques détails techniques de la mise en œuvre de VNET, puis nous présenterons les tests et les expérimentations effectués sur VNET.

Détails techniques

Comme nous l'avons déjà constaté, VNET est composé d'un client et d'un ou de plusieurs serveur(s). Ces éléments sont décrits plus en détails ci-dessous :

- **Client :** (ou utilisateur) propriétaire d'une machine virtuelle, il y accède avec sa machine physique,
- **Proxy :** machine utilisée par le client pour sa mise en réseau, en exécutant un serveur VNET. Le proxy et le client peuvent être sur la même machine,
- **Hôte :** machine physique hébergeant une ou multiples VMs,
- **Environnement local :** l'environnement local d'une VM est le LAN auquel l'hôte qui l'héberge est connecté,
- **Environnement distant :** l'environnement distant est le LAN sur lequel la machine client et la machine proxy sont connectées,

4. VTTIF est un composant de VNET permettant de déduire la topologie et la caractérisation du trafic d'une application en cours d'exécution sur les VMs.

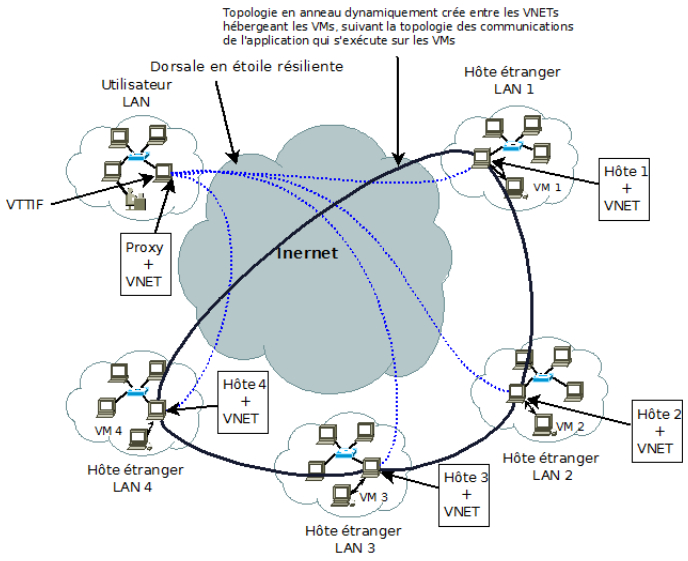


FIGURE 3.2 – VNET adapte sa topologie suivant les communications de l'application qui s'exécute sur les VMs, sans aucune intervention de l'utilisateur.

- **Session :** l'établissement d'une session avec un serveur VNET est initié par le client,
- **Handler :** après l'établissement d'une session, le client peut demander au serveur VNET d'établir un *Handler* avec un autre serveur VNET.

Une fois que VNET est téléchargé [5] et installé, nous devons lancer un serveur VNET sur chaque hôte susceptible d'instancier une VM, et sur la machine proxy, de la manière suivante :

```
vnet password [-s] -h <_|host> -p <port> -d <devices>+
```

Où :
- password : mot de passe de l'hôte correspondant,
- -s : crypter la connexion TCP (optionnel),
- -h <_|host> : adresse liée ou "_" pour toute adresse,
- -p <port> : port lié,
- -d <devices>+ : liste des interfaces Ethernet pouvant être utilisées pour le *proxing*.

Nous lançons ensuite un client VNET, sur la machine physique du client, de la manière suivante :

```
vnetclient [-s] [-p password1] [-o port1] -h <host1> -c <command>
```

Où :
- -s : crypter la connexion TCP (optionnel),
- -p password1 : mot de passe du serveur VNET sur lequel le client est connecté (proxy),
- -o <port1> : port du proxy,
- -h <host1> : le nom du proxy,
- -c <command> : la commande qui doit être faite par le proxy.

5. Le code source de VNET est disponible sur : http ://virtuoso.cs.northwestern.edu

La commande peut être une parmi les suivantes :

handle créer un handler (en ajoutant d'autres paramètres),
handlers obtenir la liste des handlers actifs,
devices obtenir la liste des interfaces du serveur VNET,
close -i <pid> fermer un handler existant,
echo ouvrir une session interactive avec le serveur VNET, où nous pouvons effectuer l'ensemble des commandes citées ci-dessus.

Expérimentations

La première expérimentation faite sur VNET consistait à héberger des VMs dans des réseaux différents et lancer un serveur VNET sur chaque hôte. Puis, lancer un client VNET pour établir une session avec un des hôtes et créer par la suite un handler avec tous les autres serveurs. En associant une adresse IP à chaque VM (des adresses d'un même réseau local), les VMs étaient capables de se *pinguer* comme si elles étaient réellement sur le même réseau physique.

Dans la deuxième expérimentation, nous avons utilisé 3 nœuds physiques appartenant à des réseaux différents. Nous avons lancé un serveur VNET et hébergé une machine virtuelle sur chacun d'eux. Nous avons utilisé ensuite un quatrième nœud pour lancer le client VNET, et configuré le réseau virtuel VNET comme le montre la figure 3.3.

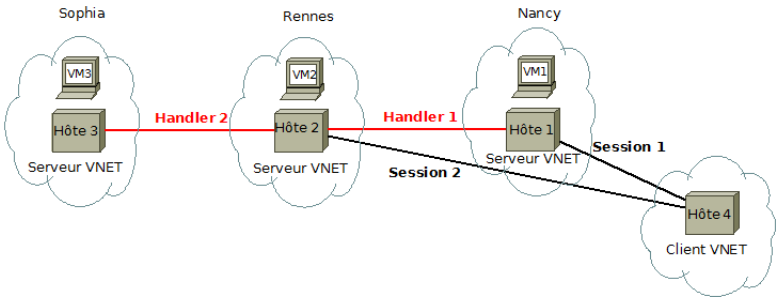


FIGURE 3.3 – Configuration de VNET

Dans cette expérimentation nous avons utilisé 3 sites de Grid'5000 : les sites de Nancy, Rennes et Sophia. Le client VNET crée une première session avec le serveur VNET de Nancy qui héberge la VM1. Il lui demande ensuite d'établir un handler avec le serveur VNET de Rennes qui héberge la VM2. Dans ce cas VM1 et VM2 se voient comme étant sur le même réseau local. Sans fermer la première session, le client VNET crée une deuxième session avec le serveur VNET de Rennes et lui demande d'établir un handler avec le serveur VNET de Sophia. Les VMs 2 et 3 se voient également comme étant sur le même LAN. Par contre VM1 et VM3 ne se voient pas dans le même réseau et ne peuvent donc pas se pinguer. À partir de ces résultats nous avons conclu que le réseau virtuel VNET ne permet pas une très grande flexibilité.

3.2 IPOP

3.2.1 Architecture

IPOP [33, 19] est un réseau virtuel de couche 3. Il permet l'ajout et la suppression dynamique des ressources sans que cela influence sur le reste du système. Son architecture se base sur deux principaux composants (cf. figure 3.4) : une interface réseau virtuelle *tap* pour capturer et injecter les paquets IP dans le réseau virtuel, et un réseau pair-à-pair *Brunet* [34] pour gérer les connexions et les mécanismes de routage. Une application IP donnée (App) se connecte à IPOP à travers l'interface réseau virtuelle. Le nœud source extrait le paquet IP de l'interface virtuelle de l'expéditeur (nœud X, gauche), et l'achemine dans l'overlay au nœud de destination. À la réception du paquet IP par le nœud destinataire (nœud Y, droite), il est injecté dans l'interface réseau virtuelle correspondante.

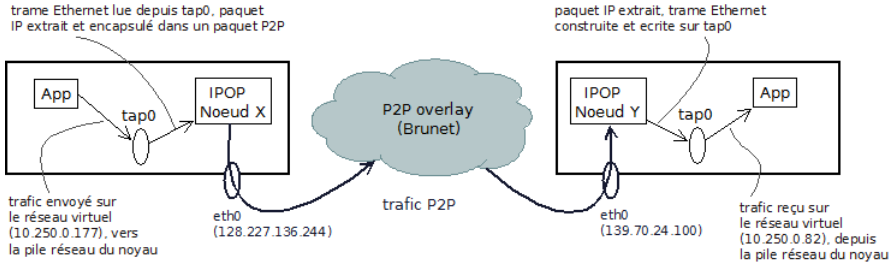


FIGURE 3.4 – Aperçu de l'architecture IPOP

IPOP permet à plusieurs réseaux IP virtuels de partager le même système pair-à-pair en utilisant un espace de nom pour chaque réseau virtuel. Il utilise également une méthode dynamique et distribuée pour l'allocation d'adresses IP en utilisant le protocole DHCP [6]. L'allocation distribuée d'adresses IP et la recherche des adresses P2P nécessite l'accès à un système de partage de données distribué, pour cela IPOP utilise la DHT produite par le système pair-à-pair sous-jacent. L'espace de nom et l'adresse IP virtuelle servent à créer une clé pour la recherche de l'adresse P2P correspondante.

Un nœud IPOP utilise l'adresse P2P, qui est découplée de l'adresses réseau physique de l'hôte, pour les messages de routage et l'adresse IP pour rejoindre le réseau overlay. En plus de l'isolation, la sécurisation et la portabilité dans IPOP, il y a également la possibilité de migrer une machine virtuelle avec la pile entière de ses logiciels d'un hôte physique à un autre. La migration d'adresse IP virtuelle arrive d'une façon transparente pour l'application. La connectivité peut être établie juste après l'établissement des liens P2P. L'auto configuration du système est transparente pour l'environnement où il réside et ne nécessite aucune interaction de l'administrateur (sauf le lancement initial).

3.2.2 Évaluation

Dans cette section, nous présentons quelques détails techniques de la mise en œuvre de IPOP. Nous montrons ensuite les expérimentations effectuées sur IPOP.

6. DHCP (Dynamic Host Configuration Protocol) est un protocole permettant d'attribuer dynamiquement des adresses IP à chaque poste de travail.

Détails techniques

Un nœud physique souhaitant rejoindre le système IPOP [7] doit nécessairement lancer une instance du nœud IPOP. Le lancement d'un nœud IPOP exige deux fichiers de configuration :

```
mono IpopNode.exe local.config ipop.config
```

- **local.config :** ce fichier concerne la configuration du nœud local. Il contient l'espace de nom du réseau virtuel correspondant, l'adresse P2P du nœud, le type de connexion (tcp, udp), etc.
- **ipop.config :** ce fichier concerne le nœud IPOP. Il contient l'espace de nom du réseau virtuel correspondant, le nom de l'interface réseau virtuelle, le nom de l'hôte IPOP, etc.

Une fois les nœuds IPOP lancés avec les fichiers de configuration correspondants, il faudra insérer ensuite l'espace de nom du réseau virtuel dans la DHT de la manière suivante :

```
python bput.py --input=dht.dhcp.namespace dhcp:namespace
```

Où :

- **dht.dhcp.namespace :** est un fichier de configuration considéré comme un serveur DHCP. Il contient principalement l'espace de nom du réseau virtuel correspondant (l'espace de nom du réseau virtuel doit être le même dans les trois fichiers de configuration) et ses adresses IP réservées.
- **dhcp :namespace** : "namespace" est l'espace de nom du réseau virtuel correspondant.

Nous pouvons voire le nombre de nœuds connectés à un réseau virtuel à travers la commande suivante :

```
python crawl.py <--debug>
```

Où –**debug** permet de voir les adresses P2P de chaque nœud.

Expérimentations

Plusieurs expérimentations ont été effectuées sur le réseau virtuel IPOP. Au début, nous avons lancé IPOP dans différents nœuds physiques appartenant à différents réseaux. Nous avons hébergé ensuite une VM sur chaque nœud. IPOP attribue automatiquement une adresse IP à chaque VM. Tout comme sur VNET les VMs étaient capables de se *pinguer* comme si elles étaient réellement sur le même réseau physique.

Nous avons essayé ensuite d'ajouter et de supprimer des nœuds du réseau virtuel IPOP déjà établi. Ces opération se sont effectué d'une façon transparente, sauf que pour l'ajout, il fallait attendre quelques secondes.

Nous nous sommes intéressé aussi à mesurer l'impact de IPOP sur les performances. Pour cela nous avons lancé l'exécution d'une application distribuée [8] sur les VMs avec et sans IPOP. Le temps d'exécution de l'application avec 4 VMs sur le même nœud physique et sur 4 nœuds physiques du même réseau était respectivement de 8% et 38% plus lent.

Le but de la dernière expérimentation était de tenter la migration d'une machine virtuelle lors de l'exécution d'une application distribuée. Nous avons migrer dans un premier temps une VM

7. Le code source de IPOP est disponible sur : http ://www.grid-appliance.org/wiki/index.php/IPOP
8. bt.A.4 des NAS Parallel Benchmarks

à un nœud physique appartenant au même réseau que le nœud source. Le temps d'exécution de l'application était plus lent d'environ 4% par rapport au temps d'exécution sans migration. Mais le plus intéressant est de migrer une VM à un nœud physique distant (appartenant à un réseau physique différent). Après avoir réaliser ce test nous avons remarqué que l'application s'est bloquée à cause d'une perte de communication entre la VM migrée et les autres VMs. Ceci est du au fait que la version d'IPOP mise en ligne ne correspond pas à celle décrite dans l'article [19] et ne contient pas le support de la migration.

Chapitre 4

Système de migration dynamique

Ce chapitre est consacré aux travaux de recherche menés durant ce stage. La première partie présente l'intérêt de notre étude et son contexte. La deuxième partie s'intéresse à la proposition d'un modèle de migration dynamique de machines virtuelles. Dans la troisième partie nous décrivons la mise en œuvre réalisée pendant ce stage. Enfin, la dernière partie s'attache à l'évaluation de notre prototype. L'évaluation a été faite sur la grille expérimentale Grid'5000.

4.1 Intérêts et contexte

La technologie des grilles de calcul et de cloud computing prend de plus en plus place dans la communauté scientifique, tout comme dans l'industrie. De nombreux axes de recherche s'intéressent à la gestion efficace de l'exécution d'applications dans ces environnements. Dans ce chapitre nous nous intéressons à la migration de machines virtuelles comme fondement de la gestion des applications réparties exécutées sur des infrastructures distribuées de type grille ou cloud. Ce problème peut être étudié de deux points de vue différents : utilisateur ou administrateur.

L'utilisateur s'intéresse principalement à réduire le coût et le temps d'exécution de ses applications. Pour cella il doit prendre en compte les schémas de communication, la disponibilité des ressources et les prix d'hébergement et de transmission de données dans les clouds qui peuvent varier en permanence.

L'administrateur quant à lui, s'intéresse à optimiser l'utilisation de ses ressources, ce qui lui permet d'économiser l'énergie ou d'avoir plus de ressources disponibles pour ses utilisateurs. Pour cella, il doit connaître l'emplacement des ressources libres et celles utilisées ainsi que le besoin des applications en cours.

Plus formellement, nous considérons trois possibilités d'optimisation :

- réduire le coût d'exécution d'une application distribuée,
- réduire le temps d'exécution d'une application distribuée,
- optimiser l'utilisation des ressources de différentes infrastructures ;

Souvent l'atteinte d'un objectif parmi les trois cités ci-dessus défavorise les deux autres.

Dans la suite de ce rapport nous nous intéressons principalement aux objectifs de l'utilisateur car ceux de l'administrateur doivent également prendre en compte d'autres contraintes telles que l'équité entre les utilisateurs. Dans le contexte des objectifs administrateur, il se peut également que l'administrateur n'ait pas une vue complète sur l'application distribuée si l'utilisateur à choisi d'utiliser plusieurs infrastructures concurrentes.

4.2 Modèle proposé

La proposition d'un système de migration dynamique de machines virtuelles est l'objectif principal de nos travaux. Dans cette section nous proposons un modèle susceptible d'assurer cette fonctionnalité. La figure 4.1 montre l'architecture de notre modèle de migration dynamique.

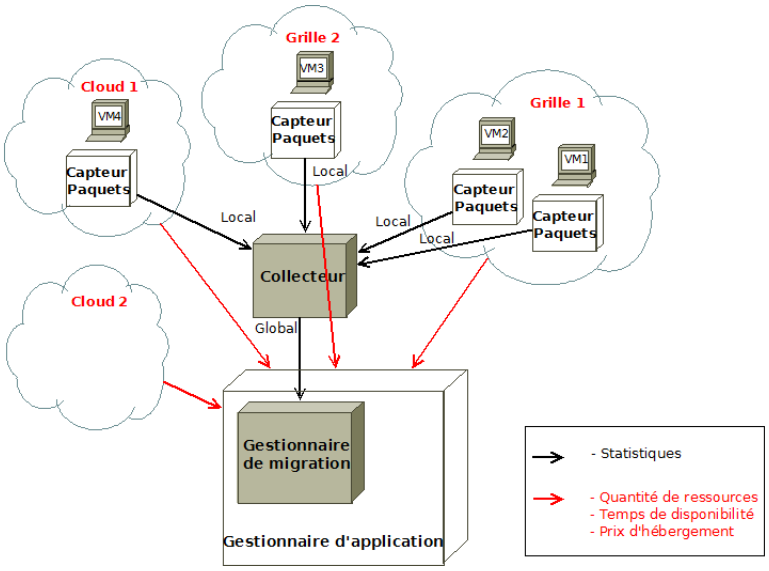


FIGURE 4.1 – Architecture de notre modèle de migration dynamique

Initialement nous considérons disposer d'une grappe virtuelle (ensemble de VMs exécutant une application distribuée). Pour chaque VM de cette grappe nous lançons un *Capteur de paquets* qui capte tous les paquets reçus par celle-ci. Dans le cas où les VMs utilisent un serveur de fichiers, le Capteur de paquets capte également les paquets envoyés à ce dernier. L'ensemble des Capteurs de paquets envoient régulièrement des statistiques à un serveur central. Le rôle du serveur central, appelé *Collecteur*, est de collecter toutes les statistiques afin d'obtenir une vue globale.

Le *Gestionnaire de migration*, un sous système du *Gestionnaire d'application*, prend les décisions de migration en se basant sur deux paramètres :

- détection de schémas de communication des VMs, à travers un mécanisme de requête/réponse avec le Collecteur.
- analyse du changement d'environnement afin de détecter d'éventuelle disponibilité de ressources plus puissantes et/ou moins chères.

Le Gestionnaire d'application s'attache à attribuer initialement chaque VM à l'environnement correspondant.

Dans ce qui suit, nous détaillerons chacun de ces éléments.

4.2.1 Capteur de paquets

Le module *Capteur de paquets* s'exécute sur chaque hôte hébergeant une machine virtuelle de la grappe virtuelle, où l'hôte lance une instance de ce module par VM. Son rôle est d'écouter le trafic de l'interface réseau virtuelle correspondante. Ce module a deux principales fonctionnalités, la première est de capturer tous les paquets envoyés à la machine virtuelle, et calculer le nombre de

messages et la taille des données envoyés par chaque adresse. La deuxième fonctionnalité consiste à envoyer périodiquement ces statistiques au module *Collecteur*. À chaque fois que le Capteur de paquets envoie les statistiques locales de sa machine virtuelle il se met à capter et calculer à nouveau sans prendre en compte les valeurs précédentes.

4.2.2 Collecteur

Le module *Collecteur* reçoit périodiquement des statistiques locales de chaque Capteur de paquets. Il les organise pour avoir des statistiques globales de tout le système. Ce module garde la trace des statistiques pour une durée maximale fixée par l'utilisateur (la durée peut être illimitée). Ceci permet au gestionnaire de migration de demander au collecteur les statistiques globales de la durée qu'il souhaite.

4.2.3 Gestionnaire de migration

Le module *Gestionnaire de migration* fait partie du module *Gestionnaire d'application*. Il est constitué de deux parties, la première consiste à analyser le changement des environnements qu'il peut utiliser. Pour cela il reçoit régulièrement les mises à jour des changements correspondants (nous supposons que ce service est offert par les environnements utilisés). La deuxième partie s'attache à la détection de schémas de communication. Pour cela le Gestionnaire de migration envoie régulièrement des requêtes au Collecteur pour avoir les statistiques concernant les communications entre machines virtuelles afin de les analyser.

Si l'objectif est de réduire le temps d'exécution sans prise en compte du coût associé, il faut détecter les machines virtuelles qui forment un groupe de communication et n'appartiennent pas au même réseau ; et détecter la disponibilité des ressources plus puissantes, pour déduire quelle stratégie adopter. Une meilleure stratégie n'est pas nécessairement de mettre toutes les VMs d'un groupe de communication dans un même réseau. En effet nous pouvons trouver un cas où il y a suffisamment de ressources disponibles dans un réseau donné, mais ces ressources ne sont pas assez puissantes par rapport à l'emplacement d'origine des VMs ; par contre la latence entre les différents réseaux est négligeable. Dans cette situation, garder les VMs là où elles sont donne un meilleur résultat pour le temps d'exécution de l'application.

Le but de minimiser le coût d'utilisation des ressources est plus complexe. En plus des schémas de communication et disponibilité et puissance des ressources, nous devons prendre en compte la variation des prix d'hébergement et d'échange de données entre clouds ainsi que les quotas des grilles. Dans le cas où nous avons la possibilité de migrer une VM d'un cloud à une grille de calcul, le coût sera certainement inférieur. Par contre il faut vérifier si le temps de disponibilité des ressources de la grille est suffisant pour exécuter l'application jusqu'à sa fin, autrement nous sommes obligés de migrer à nouveau cette VM, ce qui peut pénaliser le coût. Le temps de disponibilité des ressources dans un cloud est illimité, mais le fait de migrer une VM dans un cloud moins cher n'est pas nécessairement une bonne solution non plus. Car si les ressources du cloud moins cher sont moins puissantes, cela influencera le temps d'exécution de l'application, ce qui peut rendre le coût désavantageux.

Nous remarquons que le temps d'exécution d'une application distribuée est très important pour pouvoir prendre une décision de migration, mais il est difficile de le prédire car il dépend de la capacités des ressources et de la latence des communications entre les différentes entités de l'application.

Dans la suite de ce rapport nous supposons disposer d'un module permettant d'effectuer des calculs approximatifs du temps d'exécution d'une application donnée.

4.2.4 Gestionnaire d'application

Le rôle principale du module *Gestionnaire d'application* est d'attribuer initialement chaque VM à l'environnement correspondant (grille ou cloud), suivant les informations reçues de ces environnement (quantité de ressources disponibles, temps de disponibilité, prix d'hébergement) et suivant l'objectif principal (réduire le temps d'exécution, réduire le coût d'utilisation ou optimiser l'utilisation des ressources) de l'utilisateur.

4.3 Mise en œuvre

Durant ce stage nous avons mis en œuvre les modules *Capteur de paquets*, *Collecteur* et la partie du module *Gestionnaire de migration* qui consiste à détecter les schémas de communication. La mise en œuvre a été écrite avec le langage de programmation Java.

Dans ce qui suit nous présentons les détails techniques de notre mise en œuvre.

4.3.1 Capteur de paquets

La figure 4.2 montre le schéma du module *Capteur de paquets*. Ce module a besoin de connaître l'adresse IP du Collecteur et l'adresse MAC de la machine virtuelle correspondante (et éventuellement l'adresse IP du serveur de fichiers). La capture de paquets se fait régulièrement (toute les 30 secondes par exemple). Nous avons choisi 30 secondes comme période de temps minimale puisque, selon nous, le système ne peux pas prendre une décision de migration en se basant sur une période inférieure à celle-là.

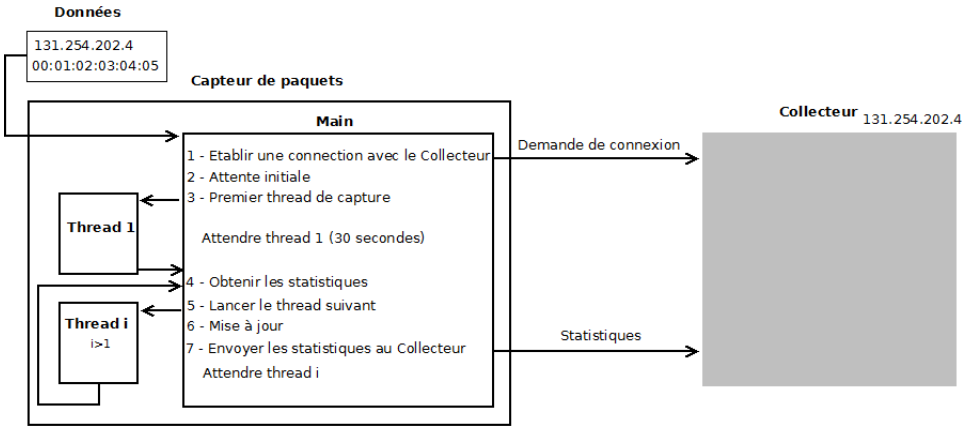


FIGURE 4.2 – Schéma du module Capteur de paquets

La fonction principale *Main* crée initialement une connexion avec le module Collecteur en utilisant la librairie Apache MINA [1]. Pour que tout les Capteurs de paquets (sur différents nœuds) obtiennent des statistiques de la même période, nous avons convenu de commencer la capture de

1. http ://mina.apache.org/

paquet à un temps précis, soit hh :mm :00 ou hh :mm :30. De ce fait le Capteur de paquets doit éventuellement attendre quelques secondes avant de commencer la capture (au pire 29 secondes). Nous supposons que les horloges sont synchronisées avec un système tel que NTP *(Network Time Protocol)*.

La fonction de capture est réalisée par un fil d'exécution (ou *thread*). Il capture tous les paquets reçus par la VM à l'aide de la librairie Jpcap [2]. Une fois que les 30 secondes s'achèvent, il envoie le nombre de messages et la taille des données reçues de chaque adresse (adresses des autres machines virtuelles communiquant avec celle-là) à la fonction principale.

Après l'obtention de ces valeurs, la fonction Main lance un nouveau thread pour la capture de paquets en parallèle avec la mise à jour de variables et l'envoi des statistiques au Collecteur. Puisque la mise à jour et l'envoi de données prennent moins de 30 secondes, la fonction principale se met à attendre la terminaison du thread de capture pour effectuer à nouveau le même traitement, et ainsi de suite.

4.3.2 Collecteur

La figure 4.3 montre le schéma du module *Collecteur*. Ce module utilise également la librairie Apache MINA pour gérer ses connexions et écouter en permanence sur un port donné (nous avons choisi 8080 par défaut) les messages reçus. S'il reçoit une nouvelle demande de connexion, il l'accepte et prend en charge par la suite toutes les statistiques envoyées par celle-ci. Ceci rend le système flexible (l'ajout et la suppression de nouveaux Capteurs de paquets est possible).

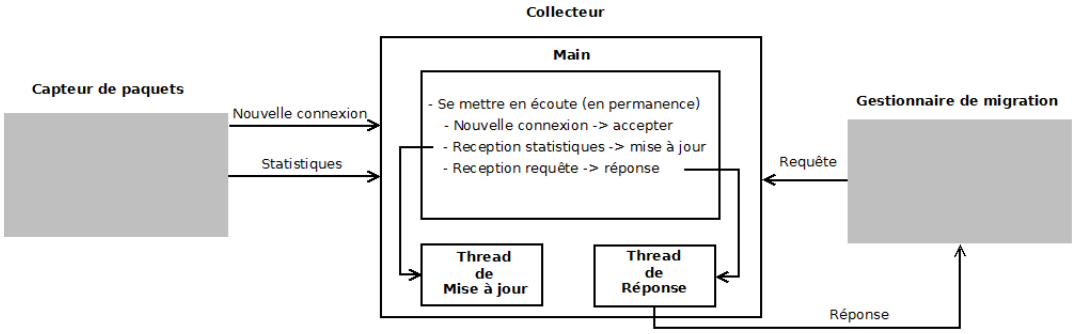


FIGURE 4.3 – Schéma du module Collecteur

À la réception de nouvelles statistiques, le Collecteur met à jour sa structure de données en mettant toutes les statistiques locales de la même période dans une même structure de données (statistiques globales de cette période). Le Collecteur garde dans sa mémoire les statistiques de plusieurs périodes, le nombre maximal de périodes est défini par l'utilisateur (par défaut illimité). De cette façon, le Gestionnaire de migration peut demander des statistiques de la durée qui lui semble adéquate.

Les requêtes envoyées par le Gestionnaire de migration comportent trois arguments : la période souhaitée, le temps d'attente maximal et l'autorisation ou non d'un résultat approximatif. Puisque les statistiques sont collectées toutes les 30 secondes, le Collecteur n'a pas en permanence les dernières statistiques. De ce fait le Collecteur doit vérifier les arguments d'une requête à sa réception. Pour que la réponse soit précise il faudrait attendre l'arrivée des dernières statistiques si le temps

2. http ://netresearch.ics.uci.edu/kfujii/jpcap/doc/

d'attente de la requête est suffisant. Autrement, si le Gestionnaire de migration indique, à travers sa requête, qu'il autorise un résultat approximatif, le Collecteur calcule approximativement les statistiques de la période partielle manquante en se basant sur celle qui la précède. Dans le cas où aucune de ces conditions n'est satisfaites, le Collecteur retourne au Gestionnaire de migration sa requête afin de lui demander de la reformuler.

Trois types de requêtes sont possibles. Le plus simple est de demander des statistiques détaillées : les valeurs correspondantes pour chaque période atomique (de 30 secondes). Le Gestionnaire de migration peut ainsi lui même calculer la moyenne suivant un vecteur de coefficient pour donner plus de priorité à certaines périodes (généralement les dernières périodes sont plus prioritaires). Le deuxième type de requête est de demander directement la moyenne des statistiques sans privilégier aucune période par rapport à une autre. Dans le troisième type il faudrait ajouter un quatrième argument à la requête qui représente un coefficient i. Dans ce cas les statistiques de la dernière période sont comptées i fois plus que celles de l'avant dernière, et les statistiques de l'avant dernière période sont comptées à leur tour i fois plus que celles de la période qui précède, et ainsi de suite. Les calculs de ces deux derniers types de requête sont faits par le module Collecteur.

4.3.3 Gestionnaire de migration

La figure 4.4 montre le schéma de la partie implémentée du module *Gestionnaire de migration*. Nous rappelons que ce module comporte deux parties principales : une partie pour la détection de groupes de communication et une partie pour l'analyse des environnements. Actuellement, les environnements tels que les grilles de calcul et les clouds n'offrent pas en permanence toutes les informations nécessaires sur la disponibilité des ressources et le coût associé. Ce qui contraint la mise en œuvre de la partie d'analyse des environnements.

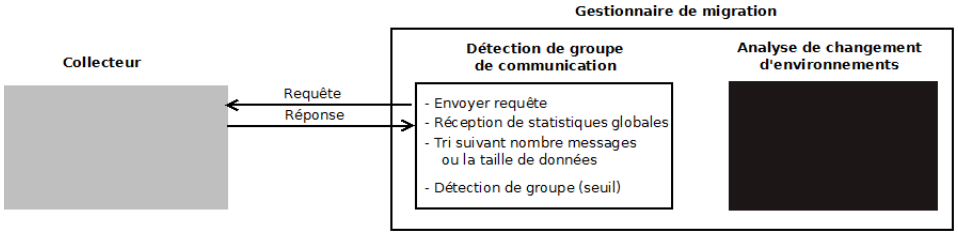


FIGURE 4.4 – Schéma partiel du module Gestionnaire de Migration

Pour la partie de détection de groupes de communication, nous utilisons la vue globale des communications entre VMs obtenue par le mécanisme de requête/réponse effectué auprès du Collecteur. Nous trions ensuite d'un ordre décroissant toutes les paires de machines virtuelles en fonction du nombre de messages échangés (ou tailles de données échangées).

Nous considérons dans un premier temps que la première paire de machines virtuelles (nombre de messages échangés le plus élevé) forme un groupe de communication. Si aucune des VMs de la seconde paire n'appartient au premier groupe alors elle forme un nouveau groupe de communication (cf. figure 4.5), et ainsi de suite.

Par contre si dans une paire donnée une des VMs appartient à un groupe existant et le nombre de messages échangés dépasse un certain seuil, nous rajoutons au groupe la VM qui ne lui appartient pas au lieu de créer un nouveau groupe (cf. figure 4.6(a)). Le seuil d'un groupe est calculé dynamiquement par rapport au nombre moyen des messages échangés entre les VMs d'un même groupe. Si le seuil

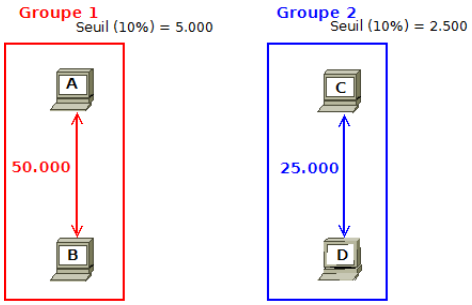
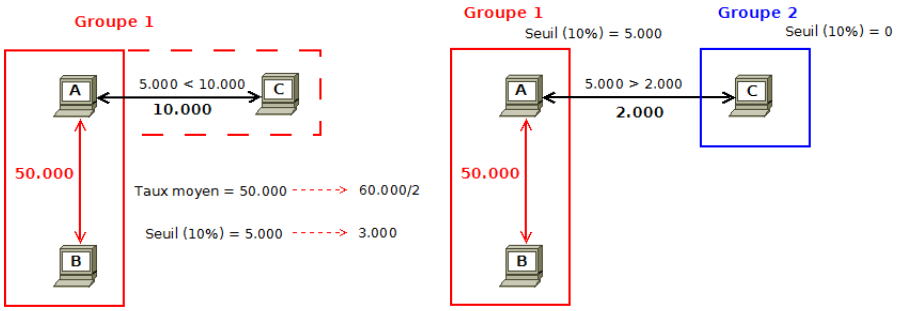


FIGURE 4.5 – Exemple avec 4 VMs

du groupe existant est supérieur au nombre de messages échangés entre la VM qui appartient au groupe et l'autre VM, alors celle-ci forme son propre groupe avec un seuil nul (cf. figure 4.6(b)).



(a) 3 VMs forment un seul groupe. (b) 3 VMs forment 2 groupes.

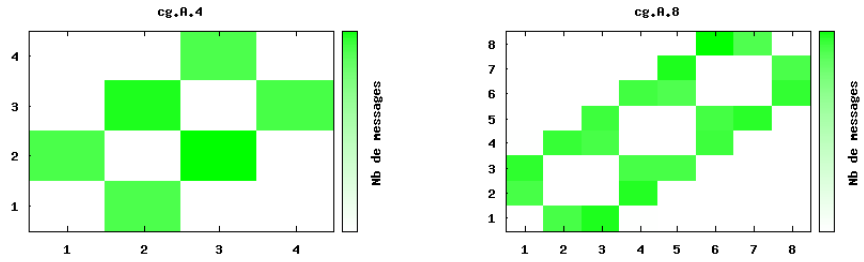
FIGURE 4.6 – Exemple avec 3 VMs

4.4 Évaluation

Nous avons évalué notre mise en œuvre sur la grille expérimentale Grid'5000. Nous avons effectué des tests avec plusieurs applications distribuées des NAS Parallel Benchmarks. Les statistiques globales ont été enregistrées par le Collecteur dans un fichier XML. À partir de ce fichier nous avons dressé des graphes représentatifs des schémas de communication de chaque application en utilisant l'outil *gnuplot* sous Linux.

Les processus des applications NAS Parallel Benchmarks communiquent entre eux à travers la librairie MPI. Le parallélisme d'une application MPI est représenté par le nombre de processus qui l'exécutent. Dans nos expérimentations chaque machine virtuelle exécute un seul processus.

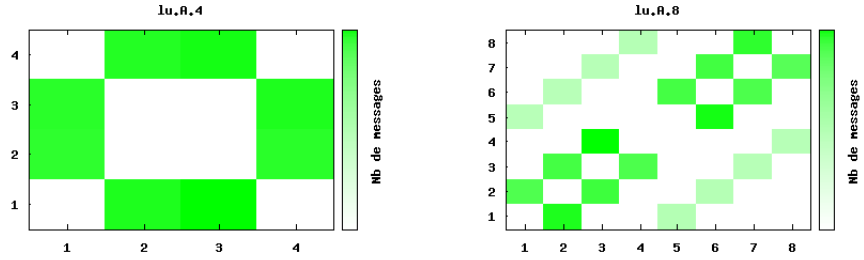
La figure 4.7 représente les schémas de communication de l'application CG pour 4 machines virtuelles (cf. figure 4.7(a)) et pour 8 machines virtuelles (cf. figure 4.7(b)). Dans le schéma de communication de l'application cg.A.4, nous remarquons que la VM1 ne communique qu'avec la VM2, et la VM4 ne communique qu'avec la VM3. Par contre VM2 et VM3 communiquent avec (VM1, VM3) et (VM2, VM4) respectivement. Nous remarquons aussi que dans le schéma de l'application cg.A.8, il y a deux groupes de communication, le groupe 1 de VM1 à VM4 et le groupe 2 de VM5 à VM8, sauf quelques communications entre VM3, VM5 et VM4, VM6.



(a) Pour 4 VMs (b) Pour 8 VMs

FIGURE 4.7 – Schéma de communication pour l'application CG

La figure 4.8 représente les schémas de communication de l'application LU pour 4 machines virtuelles (cf. figure 4.8(a)) et pour 8 machines virtuelles (cf. figure 4.8(b)). Dans l'application lu.A.4, nous remarquons que les communications sont bien réparties où chaque VM communique avec deux autres VMs. Dans l'application lu.A.8 les communications forment deux groupes (VM1 à VM4 et VM5 à VM8) qui communiquent fortement, plus quelques communications négligeables entre deux VMs de chaque groupe.

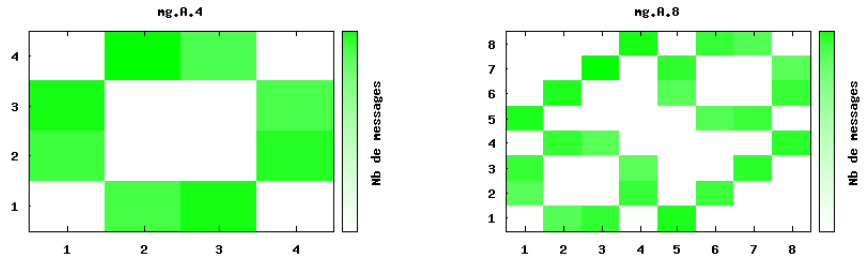


(a) Pour 4 VMs (b) Pour 8 VMs

FIGURE 4.8 – Schéma de communication pour l'application LU.

La figure 4.9 représente les schémas de communication de l'application MG pour 4 machines virtuelles (cf. figure 4.9(a)) et pour 8 machines virtuelles (cf. figure 4.9(b)) . Le schéma de communication de l'application mg.A.4 ressemble à celui de l'application lu.A.4. Les groupes de communication de l'application mg.A.8 sont les mêmes que ceux de l'application lu.A.8 sauf que les schémas à l'intérieur des groupes ne sont pas les mêmes et la communications entre les VMs de chaque groupe sont plus importantes. Dans ce cas il est plus judicieux de considérer toutes les VMs comme un seul groupe.

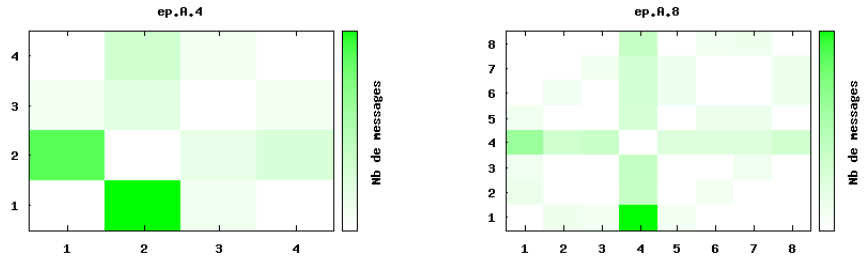
La figure 4.10 représente les schémas de communication de l'application EP pour 4 machines virtuelles (cf. figure 4.10(a)) et pour 8 machines virtuelles (cf. figure 4.10(b)). Dans les deux applications ep.A.4 et ep.A.8 les communications entre les VMs ne sont pas très importantes sauf entre VM1 et VM2 pour ep.A.4 et entre VM1 et VM4 pour ep.A.8. Ces communications ne forment pas



(a) Pour 4 VMs (b) Pour 8 VMs

FIGURE 4.9 – Schéma de communication pour l'application MG

réellement un groupe de communication.



(a) Pour 4 VMs (b) Pour 8 VMs

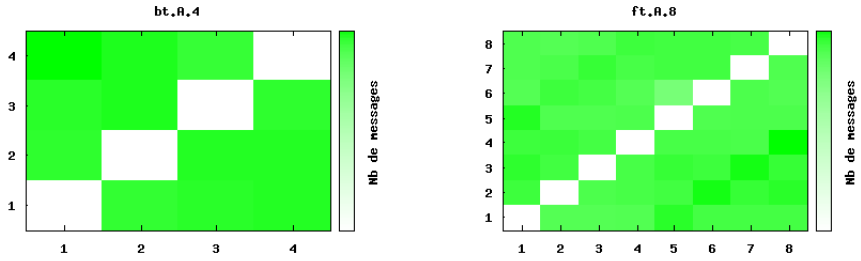
FIGURE 4.10 – Schéma de communication pour l'application EP

Les graphes de la figure 4.11 représentent le schéma de communication de l'application BT pour 4 machines virtuelles et de l'application FT pour 8 machines virtuelles respectivement. Nous avons remarqué que les schémas de communication de ces deux applications sont similaires où toutes les VMs communiquent avec toutes les VMs. Pour un schéma de communication pareil, il est fortement recommandé de positionner toutes les VMs dans la même infrastructure.

En comparant ces résultats avec les travaux précédents de détection des schémas de communication, notamment avec les résultats de [28], nous nous apercevons que nous avons obtenu les mêmes résultats pour les applications utilisant 4 machines virtuelles [3]. Ceci nous permet de valider la fiabilité de notre approche.

Afin de mesurer l'impact de notre système sur les performances (en temps d'exécution), nous avons comparé le temps d'exécution de différentes applications avec et sans utilisation de ce prototype. La différence de temps d'exécution entre les deux cas (avec et sans notre système) varie suivant la quantité de données échangées entre les entités de l'application distribuée, tel que le montre la figure 4.12 et la figure 4.13. La latence moyenne est de 2.38% pour les applications distribuées qui nécessitent 4 machines virtuelles, et de 2% pour celles qui nécessitent 8 machines virtuelles.

3. Les travaux précédents n'ont pas fait de tests sur les applications qui utilisent 8 machines virtuelles.



(a) Pour 4 VMs (b) Pour 8 VMs

FIGURE 4.11 – Schéma de communication pour l'application BT et FT

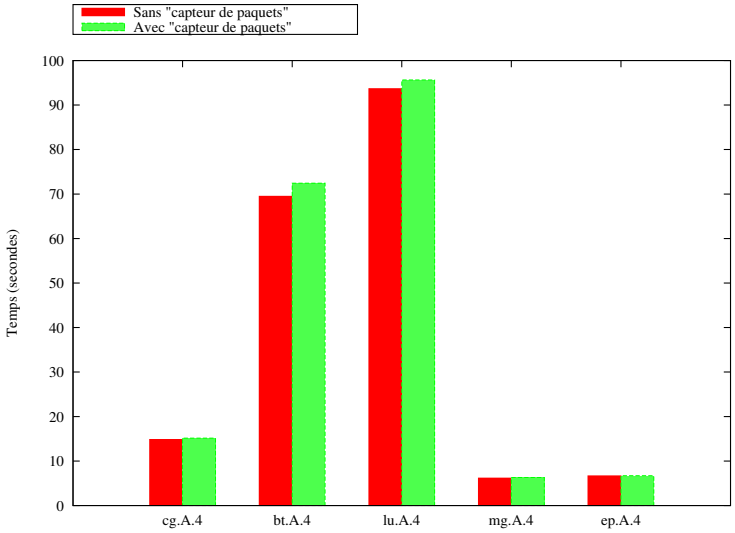


FIGURE 4.12 – Comparaison du temps d'exécution d'applications distribuées - avec et sans Capteur de paquets - pour 4 machines virtuelles.

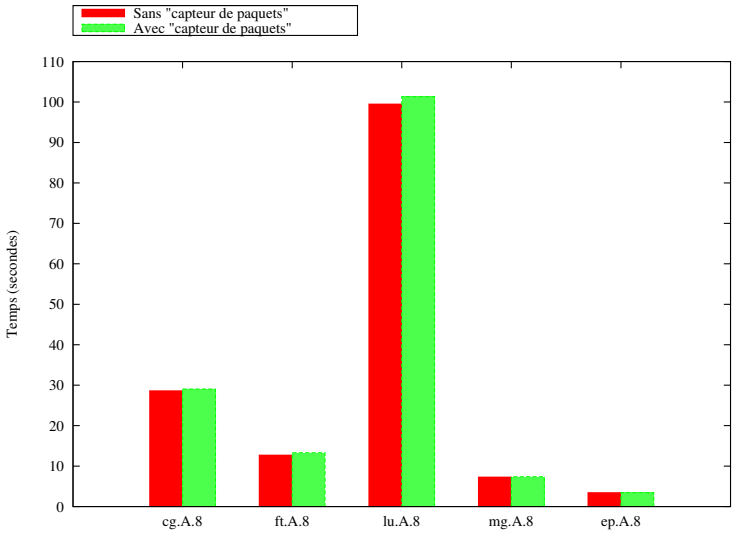


FIGURE 4.13 – Comparaison du temps d'exécution d'applications distribuées - avec et sans Capteur de paquets - pour 8 machines virtuelles.

Conclusion

La gestion transparente d'applications distribuées virtualisées dans les fédérations d'infrastructures distribuées (telles que les clouds et les grilles de calcul) est d'une importance cruciale. Bien que de nombreux mécanismes permettant de résoudre des sous parties de cette étude aient été mis en œuvre, l'étude reste incomplète et représente un sujet de recherche actuel. Dans ce rapport, nos études se sont concentrés sur la migration dynamique tenant compte des schémas de communication des machines virtuelles qui exécutent une application distribuée. Les travaux réalisés pendant ce stage de Master Recherche ont été structurés principalement en deux parties.

La première est l'évaluation du support de migration des mises en œuvres existantes de réseau virtuel. Ces mises en œuvre servent à tester notre modèle de migration dynamique des machines virtuelles. Ils étaient censés supporter la migration dynamique inter-site, mais nos tests ont montré leur inaptitude. Cela s'explique par le fait que les mises en œuvre disponibles ne correspondent pas aux mécanismes décrits dans les publications relatives.

La deuxième partie s'est attachée à la conception d'un système de migration dynamique basé sur les schémas de communication et la disponibilité et les caractéristiques des ressources. Nous avons convenu que la migration dynamique de machines virtuelles s'effectue afin d'atteindre un des 3 objectifs : minimiser le coût d'exécution, réduire le temps d'exécution ou optimiser l'utilisation des ressources physiques. Les deux premiers objectifs dépendent largement des schémas de communication, ce qui explique notre intérêt pour un système permettant de détecter ces schémas.

Différents mécanismes ont été implémentés pour détecter les schémas de communication, mais la plupart d'entre eux se fondent sur des modifications ou un traçage de la librairie MPI. Notre but est de permettre une migration dynamique, transparente et indépendante du type d'application et du mécanisme de communication utilisé. Pour cela, nous avons conçu, mis en œuvre et évalué une méthode d'analyse transparente du trafic réseau pour la détection de schémas de communication. Notre mise en œuvre a été évaluée sur la grille expérimentale Grid'5000. Nous avons obtenu des résultats similaires à ceux des travaux précédents, ce qui nous a permis de valider la fiabilité du mécanisme utilisé. L'impact de notre implémentation sur les performances en terme de temps d'exécution est faible (environ 2%).

Néanmoins, une optimisation de notre mise en œuvre est en cours, afin d'obtenir de meilleures performances. Il nous semble avantageux d'étudier la possibilité de lancer un seul Capteur de paquets sur une même machine physique quelque soit le nombre de machines virtuelles hébergées par celle-ci. Il faudrait également mettre en œuvre le module Gestionnaire d'application qui doit placer chaque machine virtuelle dans l'environnement adéquat au lancement de l'application distribuée, et implémenter aussi la partie manquante du Gestionnaire de migration qui consiste à analyser le changement des environnements.

Enfin, suite à notre étude effectuée sur les réseaux virtuels, il apparait qu'il n'existe pas de mise

en œuvre d'un tel mécanisme permettant la migration inter-site transparente. Ceci est un point majeur qui reste à traiter pour finaliser notre étude.

Bibliographie

[1] James E. Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5) :32–38, 2005.

[2] Tim Lindholm and Frank Yellin. *Java Virtual Machine Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] R. P. Goldberg. Architecture of virtual machines. In *Proceedings of the workshop on virtual computer systems*, pages 74–112, New York, NY, USA, 1973. ACM.

[4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03 : Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[5] http ://www.vmware.com/.

[6] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm : the Linux Virtual Machine Monitor. In *Proceedings of the 2007 Linux Symposium*, volume 1, pages 225–230, June 2007.

[7] http ://www.virtualbox.org/.

[8] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI'05 : Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[9] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast transparent migration for virtual machines. In *ATEC '05 : Proceedings of the annual conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2005. USENIX Association.

[10] Charles E. Perkins and David B. Johnson. Mobility support in ipv6. In *MobiCom '96 : Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 27–37, New York, NY, USA, 1996. ACM.

[11] Eric Harney, Sebastien Goasguen, Jim Martin, Mike Murphy, and Mike Westall. The efficacy of live virtual machine migrations over the internet. In *VTDC '07 : Proceedings of the 2nd international workshop on Virtualization technology in distributed computing*, pages 1–7, 2007.

[12] Ezra Silvera, Gilad Sharaby, Dean Lorenz, and Inbar Shapira. Ip mobility to support live migration of virtual machines across subnets. In *SYSTOR '09 : Proceedings of SYSTOR 2009 : The Israeli Experimental Systems Conference*, pages 1–10, New York, NY, USA, 2009. ACM.

[13] Jorge Carapinha and Javier Jiménez. Network virtualization : a view from the bottom. In *VISA '09 : Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 73–80, New York, NY, USA, 2009. ACM.

[14] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *SOSP '01 : Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 131–145, New York, NY, USA, 2001. ACM.

[15] Xuxian Jiang Dongyan. Violin : Virtual internetworking on overlay infrastructure, 2003.

[16] M. Tsugawa and J.A.B. Fortes. A virtual network (vine) architecture for grid computing. *Parallel and Distributed Processing Symposium, International*, 0 :123, 2006.

[17] Mahesh Kallahalla, Mustafa Uysal, Ram Swaminathan, David E. Lowell, Mike Wray, Tom Christian, Nigel Edwards, Chris I. Dalton, and Frederic Gittler. Softudc : A software-based data center for utility computing. *Computer*, 37(11) :38–46, 2004.

[18] Ananth I. Sundararaj, Ashish Gupta, and Peter A. Dinda. Dynamic topology adaptation of virtual networks of virtual machines. In *LCR '04 : Proceedings of the 7th workshop on Workshop on languages, compilers, and run-time support for scalable systems*, pages 1–8, New York, NY, USA, 2004. ACM.

[19] David Isaac Wolinsky, Yonggang Liu, Pierre St. Juste, Girish Venkatasubramanian, and Renato Figueiredo. On the design of scalable, self-configuring virtual networks. In *SC '09 : Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM.

[20] Ala Rezmerita, Tangui Morlier, Vincent Néri, and Franck Cappello. Private virtual cluster : Infrastructure and protocol for instant grids. In *Euro-Par 2006 Parallel Processing*, pages 393–404, 2006.

[21] Luca Deri and Richard Andrews. N2n : A layer two peer-to-peer vpn. In *AIMS '08 : Proceedings of the 2nd international conference on Autonomous Infrastructure, Management and Security*, pages 53–64, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] Dilip Joseph, Jayanth Kannan, Ayumu Kubota, Karthik Lakshminarayanan, Ion Stoica, and Klaus Wehrle. Ocala : An architecture for supporting legacy applications over overlays. In *In NSDI*, 2006.

[23] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks—summary and preliminary results. In *Supercomputing '91 : Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 158–165, New York, NY, USA, 1991. ACM.

[24] Rolf Riesen. Communication patterns [message-passing patterns]. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.

[25] I. Lee. Characterizing communication patterns of nas-mpi benchmark programs. Southeast Conference, Atlanta, 2009.

[26] Robert Preissl, Thomas Köckerbauer, Martin Schulz, Dieter Kranzlmüller, Bronis R. de Supinski, and Daniel J. Quinlan. Detecting patterns in mpi communication traces. In *ICPP '08 : Proceedings of the 2008 37th International Conference on Parallel Processing*, pages 230–237, Washington, DC, USA, 2008. IEEE Computer Society.

[27] Robert Preissl, Martin Schulz, Dieter Kranzlmüller, Bronis R. Supinski, and Daniel J. Quinlan. Using mpi communication patterns to guide source code transformations. In *ICCS '08 : Proceedings of the 8th international conference on Computational Science, Part III*, pages 253–260, Berlin, Heidelberg, 2008. Springer-Verlag.

[28] Srikanth Goteti and Jaspal Subhlok. Communication pattern based node selection for shared networks. In *In Autonomic Computing Workshop : The Fifth Annual International Workshop on Active Middleware Services (AMS 2003*, 2003.

[29] Amitoj Singh and Jaspal Subhlok. Reconstruction of application layer message sequences by network monitoring, 2002.

[30] https ://www.grid5000.fr.

[31] Ananth I. Sundararaj and Peter A. Dinda. Towards virtual networks for virtual machine grid computing. In *VM'04 : Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*, pages 14–14, Berkeley, CA, USA, 2004. USENIX Association.

[32] Amit P Kucheria. Scalable emulation of ip networks through virtualization.

[33] Arijit Ganguly, Abhishek Agrawal, P. Oscar Boykin, and Renato J. O. Figueiredo. Ip over p2p : Enabling self-configuring virtual ip networks for grid computing. *CoRR*, abs/cs/0603087, 2006.

[34] http ://boykin.acis.ufl.edu/wiki/index.php/Brunet.