l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

ECOLE DES MINES DE NANTES

Thèse de Doctorat

Rémy Pottier

Mémoire présenté en vue de l'obtention
du grade de Docteur de l'École des Mines de Nantes
Sous le label de l'Université Nantes Angers Le Mans

Discipline : Informatique
Spécialité : Informatique
Laboratoire : École des Mines de Nantes

Soutenue le 19 septembre 2012

École doctorale : Sciences et Technologies de l'Information et des Mathématiques

Thèse N° 2012EMNA0017

Un langage dédié à l'administration d'infrastructures virtualisées

JURY

Rapporteurs :	**M. Pazat Jean-Louis**, Professeur, Institut National des Sciences Appliquées de Rennes
	M. Noël De Palma, Professeur, Université de Grenoble
Examinateurs :	**M. Jean-Claude Royer,** Professeur, École des Mines de Nantes
	M. Jean-Marc Menaud, HdR, École des Mines de Nantes
	M. Jean-Pierre Guédon, Professeur, École Polytechnique de Nantes
	M. Gaël Thomas, Maître de Conférences, Université de Paris
	M. Frédéric Dang Tran, Ingénieur de recherche, Orange Labs de Issy Les Moulineaux

VMScript, un langage dédié à l'administration d'infrastructures virtualisées

VMScript, A Domain Specific Language Ror Virtualized Infrastructures

Rémy Pottier

Université de Nantes

Remerciements

Ces trois années et demie de doctorat ont été pour moi l'occasion d'apprendre une grande quantité de connaissances scientifiques ainsi que de nombreuses autres compétences que cette seule page ne suffirait pas à détailler. Ces différents apprentissages n'auraient été possibles sans l'aide de personnes que j'aimerai maintenant remercier.

Je commencerai par remercier l'École des Mines de Nantes pour m'avoir offert le matériel, les conditions financières et un bureau propice aux réflexions intenses et à ces moments de détente qui ont été salvateurs.

Aux membres de l'équipe de recherche ASCOLA qui ont été mon premier auditoire pour la présentation des travaux menés dans le cadre de cette thèse. Vos remarques m'ont aidé à avancer et à construire mon raisonnement scientifique.

Merci aux membres de mon jury, Pazat Jean-Louis, Noël De Palma, Jean-Claude Royer, Jean-Pierre Guédon et Gaël Thomas pour avoir lu, commenté et évalué les travaux de ce doctorat.

Merci aux membres du projet SelfXL pour ces réunions riches en réflexion et cette bonne ambiance toujours présente. À Frédéric Dang Tran pour ses réflexions autour des gestionnaires de machines virtuelles et pour m'avoir aidé à prendre en main l'outil Sirocco. Merci également à Bull, et particulièrement à Benoît Pelletier et Christian Bourgeois et leur équipe, pour m'avoir accueilli dans leurs locaux et m'avoir fait profiter d'une expertise professionnelle qui m'a permis d'avancer dans mes réflexions.

Un énorme merci à mon encadrant de thèse, Jean-Marc Menaud pour m'avoir guidé avec talent depuis mon stage de fin de master jusqu'à ma soutenance de thèse. Merci pour m'avoir donné l'opportunité d'exprimer mes idées, pour ce temps précieux que tu as réussi à me libérer dans ton emploi du temps plus que chargé, pour ton engagement dans la recherche informatique en général et dans nos travaux de recherche en particulier et aussi pour m'avoir apporté bien plus que des connaissances techniques.

Je souhaite aussi remercier, Adrien Lèbre, pour les nombreuses discussions que nous avons eu, ses traits d'humour toujours bien placés et nos travaux en commun, en particulier pour la démonstration d'Entropy qui fait partie de ces moments que je ne serai oublié. Merci pour ta passion et ta bonne humeur.

Merci à Marc Léger pour m'avoir offert une année de travail, une publication et une expertise qui m'aurait fait défaut. J'ai beaucoup appris en peu de temps. Sans toi, cette thèse n'aurait pas été la même.

Bien sûr, je remercie Flavien Quesnel, pour ses nombreuses discussions sans fin qui ont nourri mes travaux de recherche pendant toutes ces années. Pour avoir partagé nos moments de doute, de très mauvaises blagues (mais aussi quelques bonnes), notre avenir de docteur, nos succès et nos déceptions mais aussi pour avoir fait de la salle « Ali Assaf » l'endroit le plus convivial du département informatique.

J'en profite pour remercier mes autres collègues de bureau Frederico Álvares, Gustavo Brand et Ismael Mejìa qui ont participé à l'excellente ambiance de ce lieu d'échange et de travail.

Merci à Alexandre Garnier et Frédéric Dumont pour avoir été des stagiaires exemplaires, remplis d'humour et de bonne humeur. Vos codes ont enrichi les travaux de ce doctorat et vos études sociologiques des animaux du prés voisin m'ont souvent permis de décompresser.

Je finirai par remercier Martin Dargent et Thierry Bernard qui ont accepté de relever le défi de l'aventure EasyVirt, une start-up pleine d'avenir.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Présentation de la thèse

1.1 Problématique

L'outil informatique est de plus en plus présent dans notre vie quotidienne (internet, smartphones, jeux vidéos en ligne) comme dans les entreprises (bases de données, services web). Cependant, le principe de l'outil informatique installé sur les postes fixes des utilisateurs est entrain d'être révolutionné. La combinaison d'Internet et du Web 2.0 (client léger) a provoqué un changement radical. De nos jours, les applications deviennent de plus en plus accessibles à distance au travers d'un simple navigateur internet. L'exemple le plus populaire est, sans nul doute, celui du client mail. Actuellement, tous les fournisseurs d'accès internet (FAI) proposent à leurs clients d'accéder à leur messagerie depuis une application hébergée chez les FAI et accessible à distance. De manière générale, nous appelons service en ligne ces applications accessibles à distance via un navigateur.

À l'instar des sites web classique, ces services en ligne peuvent être hébergés en interne à la société proposant le service (comme pour les FAI), ou en externe, chez des hébergeurs. Le traitement des données associées à ce service n'est donc plus exécuter sur le poste du client mais sur les serveurs du fournisseur de services. Par conséquent, ce dernier doit s'équiper de matériels toujours plus puissants et plus nombreux pour répondre à une demande toujours en croissance. De nombreux bâtiments ont alors été construits dans le but d'entreposer un nombre important de serveurs dont le rôle est l'hébergement d'applications et le stockage des données. Ces bâtiments, appelés des centres de données, offrent les avantages suivants :

- le regroupement de matériels de qualité comme des serveurs performants, des climatiseurs, des équipements de communications réseau très haut débit, des générateurs électriques ;
- un environnement très contrôlé (la quantité de poussière et la température sont surveillées en permanence) ;
- une gestion globale d'un nombre conséquent de serveurs pouvant aller jusqu'à plusieurs milliers de serveurs ;
- une plus grande protection contre les incendies, les coupures électriques, les intrusions physiques et logicielles.

Avec l'accroissement du nombre d'applications à héberger, les centres de données sont devenus des structures complexes. Afin de minimiser le coût d'exploitation d'un centre de données, son organisation doit être étudiée de la conception du bâtiment jusqu'à la gestion des applications qu'il héberge. Un de points concerne l'énergie électrique. Les centres de données consomment énormément d'énergie et utilisent 1% de l'électricité mondiale. Des réductions de dépenses énergétiques des centres de données sont donc nécessaires [Lib10]. Pour cela, de nombreux projets comme Datadock et Big Blue prennent en compte la dimension énergétique dès la conception du

centre de données [BR10]. Une fois le centre de données construit, des économies peuvent encore être effectuées en organisant efficacement les applications hébergées par le centre de données. En effet, un matériel de qualité ne peut résoudre les problèmes liés à une mauvaise gestion des applications. Une bonne gestion des applications doit garantir un fonctionnement correct des applications tout en hébergeant un maximum d'applications sur les serveurs du centre de données. L'organisation de ces applications au sein d'un centre de données est appelée « l'administration du centre de données ». L'administration de ce type d'infrastructures est particulièrement difficile. À la croisée des chemins de différents métiers (bâtiment, réseau, flux de température, air, électricité, etc.), elle nécessite pour l'administrateur de disposer, puis de comprendre, les multiples vues de son parc de serveurs. De plus, avec l'apparition de la virtualisation, ces tâches se sont complexifiées. Cette technologie permet de regrouper plusieurs applications dans un container appelé une machine virtuelle (VM, de l'anglais « Virtual Machine »). Plusieurs VM peuvent être exécutées sur un même serveur physique. Initialement déployée pour rationaliser l'utilisation des serveurs, la virtualisation a apporté son lot de fonctionnalités censées faciliter son administration. Une machine virtuelle peut, entre autre, être déplacée d'un serveur à un autre sans arrêter les applications qu'elle exécute.

Ce besoin de consolidation est né d'un constat simple : un serveur d'un centre de données utilise en moyenne environ 20 % de ses capacités. Une meilleure gestion des applications permettrait d'héberger les applications sur moins de serveurs et donc de diminuer le nombre de serveurs à utiliser. Cependant, la répartition des applications sur les serveurs n'est pas triviale. Pour fournir un service correct, une application doit disposer de suffisamment de ressources (CPU, mémoire, réseau). Cependant, la consommation de ressources d'une application varie et est rarement prévisible. Par exemple, un site commercial connait un pic d'activité, plus ou moins fort, en dehors des horaires de travail. Classiquement, afin d'assurer un service de qualité, les ressources réservées à chaque application sont surévaluées pour parer à toute éventualité. L'amélioration du taux d'utilisation des serveurs passe par une étape de consolidation dynamique des serveurs, c'est-à-dire, être capable d'héberger plus d'applications sur un même nombre de serveurs en utilisant la virtualisation.

De plus, le taux de consolidation est actuellement de 6 VM par processeur, l'effet pervers de la virtualisation est d'avoir multiplié par 7 (les 6 VM + le serveur) le nombre d'éléments à administrer, et donc d'avoir complexifié l'administration des centres de données.

1.2 Objectifs

Les travaux de cette thèse s'intéressent à la gestion de milliers de machines virtuelles dans les centres de données virtualisés. Comme une même entité peut être amenée à gérer une fédération de centres de données, dans la suite de ce document, le terme « infrastructure » sera préféré à celui de « centre de données ». Dans une infrastructure, une gestion efficace des applications permet d'économiser de l'énergie et de fournir un service de qualité. Cependant, cette gestion ne peut être dissociée de la gestion de la couche sous-jacente, les serveurs. L'ensemble des tâches de gestion, comme l'ajout de serveurs dans une infrastructure et la configuration des serveurs et des applications, est appelé l'administration. L'objectif de cette thèse est de proposer un outil d'administration facilitant la manipulation des serveurs et des machines virtuelles. Deux étapes sont indispensables pour effectuer l'administration d'une infrastructure virtualisée. La première étape est l'obtention d'une vue de l'infrastructure dans son ensemble afin de localiser les serveurs (par ex., en cas de panne) et l'observation des machines virtuelles et de leurs ressources pour améliorer l'organisation de l'infrastructure. La deuxième étape est la reconfiguration de l'infrastructure afin de garantir son bon fonctionnement. L'outil proposé doit être modulaire

afin de s'adapter à l'organisation et à la configuration des infrastructures. Ces dernières sont de tailles très variables, d'une dizaine de serveurs à plusieurs milliers. La gestion d'un grand nombre de serveurs et de machines virtuelles est donc indispensable. Les interfaces graphiques étant inadaptées à la gestion de nombreux éléments (la sélection de mille éléments à la souris demande beaucoup de patience), l'outil proposé se doit de proposer une approche langage.

1.3 Contributions

Les contributions principales de cette thèse peuvent être classifiées en deux parties. La première partie des contributions concerne les langages développés pour la gestion des infrastructures virtualisées.

Les différentes étapes de l'administration sont abordées à partir de deux langages : un langage dédié à la description d'infrastructures virtualisées et un langage dédié à l'introspection et à la reconfiguration de l'infrastructure. L'introspection d'une infrastructure consiste à recueillir des informations sur les différents éléments comme la liste des VM hébergées par un serveur. La reconfiguration de l'infrastructure permet les opérations d'administration comme l'ajout ou l'arrêt de VM.

Une des originalités de la thèse est d'avoir retenu l'approche « langage dédié » (DSL, de l'anglais « Domain Specific Language ») pour la définition de ces langages. Un DSL est un langage de programmation dont les spécifications sont dédiées à un domaine d'application précis. Les langages dédiés s'opposent aux langages de programmation généralistes comme le langage Java. L'objectif d'un langage dédié est double. Le premier est de définir une sémantique et des abstractions langages permettant d'écrire des programmes dédiés bien plus rapidement qu'avec un langage de programmation généraliste. Pour cela, un langage dédié se base sur des librairies permettant d'adresser simplement le domaine spécifique. Au delà des abstractions, le langage définit une grammaire et donc des règles d'utilisation de la librairie. Le deuxième objectif est de proposer un système de vérification des programmes développés. La première vérification est statique et grammaticale. Elle permet de vérifier la bonne utilisation de la librairie. La seconde vérification a lieu au niveau de l'exécution des programmes. Cette vérification est effectuée dans l'environnement d'exécution et est liée au contexte d'exécution.

La deuxième partie de la contribution concerne l'environnement d'exécution des langages dédiés, sa généricité et ses capacités de passage à l'échelle. Le langage de description [PLM10] décrit une infrastructure en la séparant en plusieurs vues. Par exemple, une vue décrivant l'organisation des serveurs, une vue décrivant l'organisation de machines virtuelles et une autre vue décrivant les installations électriques. Grâce à la généricité du problème, le nombre de vues et leur structure sont définies par l'administrateur ce qui permet à l'outil de s'adapter à la taille et à la configuration des infrastructures.

Le second langage [PLM10, PM12] permet l'introspection et la reconfiguration de l'infrastructure. Un système modulaire permet l'utilisation de divers outils d'observation et de reconfiguration (Libvirt, VMware VCenter, Sirocco). Pour finir, une évaluation des langages et de leurs environnements d'exécution a été effectuée.

1.4 Organisation du document

Ce document est organisé en deux parties. La première partie décrit le contexte de la thèse. Dans le chapitre 2, les principes de la virtualisation et de sa démocratisation avec l'utilisation de plus en plus courante de l'informatique dans les nuages sont exposés. Le chapitre 3 clôture

cette partie en détaillant les différentes étapes de l'administration puis quelques uns des outils s'acquittant, en partie ou en totalité, des tâches d'administration.

La deuxième partie de ce document est consacrée aux contributions de la thèse. Le chapitre 4 présente les langages dédiés conçus pour l'administration d'infrastructures virtualisées. Le chapitre 5 détaille l'architecture et les apports du prototype implémenté. Le chapitre 6 évalue la concision et l'utilité d'un langage dédié à l'administration puis explique le déploiement de l'outil sur une infrastructure virtualisée contenant plus de deux cents serveurs. Finalement, le chapitre 7 conclue cette thèse et expose les perspectives liées à ce travail.

1.5 Diffusion scientifique

Les différents travaux présentés dans ce document ont fait l'objet de publications présentées dans des conférences internationales avec comité de lecture :

[1] Rémy Pottier, Marc Léger, Jean-Marc Menaud. A Reconfiguration Language for Virtualized Grid Infrastructures. In *DAIS'10 : Proceedings of the 10th IFIP International Conference on Distributed Applications and Iteroperable Systems*, pages 42–55, Amsterdam, Netherlands, 2010.

[2] Rémy Pottier, Jean-Marc Menaud. A Safe Management System for Virtualized Data Center. In *ICAS'12 : Proceedings of the Eighth International Conference on Autonomic and Autonomous Systems*, Sint Maarten, Netherlands, 2012.

Première partie

Contexte

Chapitre 2

La virtualisation et l'informatique dans les nuages

La taille des centres de données est en constante augmentation ce qui complexifie l'organisation des applications hébergées par ces infrastructures. Afin de rendre ces infrastructures plus flexibles, la virtualisation est utilisée pour créer, arrêter, déplacer les applications au sein d'une infrastructure dite virtualisée. Avec l'accroissement des besoins informatiques, l'informatique dans les nuages proposent aujourd'hui d'obtenir des ressources diversifiées : des emplacements pour héberger des serveurs, des serveurs pour héberger des applications, des services prêt à l'emploi. La structure d'une infrastructure virtualisée est donc complexe : de nombreux utilisateurs, des applications avec des besoins différents, des ressources hautement évolutives. Afin de s'adapter à un tel environnement et de fournir une administration efficace, des outils aidant l'administrateur sont nécessaires. Ces outils offrent des fonctionnalités d'observation des ressources, de reconfigurations de l'infrastructure et de détection de problèmes assurant la cohérence de l'infrastructure.

Sommaire

Dans les centres de données, une application utilise les ressources mises à sa disposition par le serveur l'hébergeant. Il a donc été naturel dans un premier temps d'héberger une application par serveur. Les centres de données hébergeant de plus en plus d'applications, le besoin de consolider plusieurs applications sur un même serveur s'est fait sentir afin d'éviter un accroissement incommensurable du nombre de serveurs. Cependant, l'hébergement de plusieurs applications sur un même serveur nécessite quelques précautions. En effet, si les ressources du serveur sont insuffisantes, le fonctionnement des applications hébergées sera dégradé. La consommation de ressources d'une application est dynamique, c'est-à-dire, qu'elle évolue dans le temps. Il faut donc que le serveur dispose, continuellement, de suffisamment de ressources pour toutes les applications qu'il héberge. Par exemple, la consommation d'une application web augmente avec son nombre d'utilisateurs. L'administrateur doit donc connaître la consommation maximale de l'application pour l'héberger sur un serveur sans risque de dysfonctionnement. Afin d'assurer à chaque application les ressources nécessaires, les serveurs sont généralement utilisés à 20% de leur capacité. Si la charge de toutes les applications augmentent simultanément, le serveur dispose de 80% de ces ressources pour garantir le fonctionnement correct des applications. Un exemple de montée en charge conséquent est la télédéclaration des impôts. Sur une période de quelques semaines, les serveurs de la trésorerie publique connaissent une activité importante. Pendant le reste de l'année, ces serveurs consomment moins de ressources. La réorganisation d'une infrastructure d'après les ressources consommées nécessite une administration dynamique facilitée par la virtualisation.

La virtualisation augmente la flexibilité de l'infrastructure en regroupant les applications dans des containers appelés machines virtuelles (VM, de l'anglais Virtual Machine). Cette flexibilité peut permettre de réduire le nombre de serveurs utilisés en consolidant les applications d'un centre de données. L'utilisation d'un nombre limité de serveurs entraînent des gains de place dans les centres de données et des économies d'énergie. La virtualisation facilite aussi le test d'applications. En effet, le test d'une application peut être fait rapidement en créant une VM spécialement pour le test. Une fois le test terminé, la VM est détruite.

2.1 La virtualisation

2.1.1 Définition

Le principe de la virtualisation [Gol73, SN05] consiste à exécuter plusieurs systèmes d'exploitation sur un seul serveur. Chaque système d'exploitation est isolé de ces confrères dans une VM. Chaque VM possède donc un système d'exploitation permettant d'exécuter des applications.

Combiné avec l'arrivée des architectures multiprocesseur et multicœur, l'intérêt principal de la virtualisation est de permettre une plus grande consolidation des applications. La consolidation est le regroupement cohérent de données. La consolidation d'applications sur des serveurs est définie par le regroupement de ressources sur un ensemble de serveurs. La mutualisation d'un serveur entre plusieurs applications permet de mieux utiliser le serveur. De plus, certaines applications sont validées sur des versions précises d'un système d'exploitation. Toutes les applications doivent donc être validées pour le même système d'exploitation. Au vue du nombre de versions de Linux, qui est le système d'exploitation le plus utilisé dans les centres de données, cette restriction limitait la consolidation. Sans la virtualisation, la mutualisation d'un serveur entre plusieurs applications impose l'utilisation d'un seul système d'exploitation pour l'exécution de toutes les applications du serveur. On ne peut donc pas consolider sur un même serveur deux applications utilisant des systèmes d'exploitation différents. Avec la virtualisation, chaque VM

possède son propre système d'exploitation et donc plusieurs systèmes d'exploitation peuvent être hébergés par un seul serveur.

Cependant, lors d'une consolidation, la corruption d'une application peut entrainer des dysfonctionnements ou la corruption des applications hébergées sur le même serveur. La cohabitation d'applications sur un même serveur implique un partage des ressources et des périphériques par les applications. Si deux applications sollicitent fortement le même périphérique, par exemple, le disque dur, une perte de performance sera observée.

La gestion des VM est effectuée par un hyperviseur [RG05] qui est une fonctionnalité apportée par le système d'exploitation du serveur. Il existe deux types d'hyperviseurs [MK07] :

- les hyperviseurs de type 1 (aussi appelés « natif » ou « bare metal ») font partie intégrante du système d'exploitation. C'est le cas des hyperviseurs KVM [KKL+07], Hyper-V, VMware ESXi [VMwa] et Xen [BDF+03]. Sur les trois derniers hyperviseurs cités, il peut être nécessaire de modifier le système d'exploitation de la VM si le processeur du serveur ne dispose pas des instructions de virtualisation. On parle alors de paravirtualisation ;
- les hyperviseurs de type 2 sont des applications à installer a posteriori sur un système d'exploitation. Les hyperviseurs de type 2 les plus connus sont Parallels Desktop, VirtualBox, VMware Fusion[VMwa].

Les hyperviseurs de type 2 s'installent comme une application ordinaire sur un système d'exploitation existant. Leur installation est donc simple et rapide. De plus, le système d'exploitation reste utilisable pour effectuer des tâches autres que la virtualisation.

Les hyperviseurs de type 1 nécessite l'installation d'un système d'exploitation entier. Le processus d'installation est donc plus complexe et plus long. De plus, les hyperviseurs comme VMWare et Hyper-V sont spécialisés dans la virtualisation, il n'est donc pas possible d'effectuer d'autres tâches. Les hyperviseurs de type 1 sont en général plus performant que ceux de type 2 car la communication entre l'hyperviseur et le système d'exploitation n'existe pas (l'hyperviseur est le système d'exploitation).

Le serveur hébergeant l'hyperviseur est souvent dédié à la gestion de VM. Aucune autre application comme des bases de données ou des applications web ne s'exécutent. Par abus de langage, le terme hyperviseur peut être donc utilisé pour désigner le serveur dédié à l'hébergement de VM.

Un hyperviseur permet d'éteindre, de redémarrer, de suspendre, de migrer ou de sauvegarder une VM. La suspension (ou pause) stocke la mémoire de la VM sur le disque dur de l'hyperviseur avant d'arrêter tous les calculs effectués par la VM. La VM ne consomme alors plus de ressources CPU, mémoire ou réseau. Lors de la sortie de suspension, la mémoire de la VM est réactivée et les calculs reprennent. La sauvegarde (snapshot, en anglais) d'une VM crée un fichier permettant de redémarrer la VM dans un état antérieur. La migration d'une VM permet le déplacement d'une VM d'un serveur à un autre (le processus de migration sera détaillée dans la section 2.1.3).

2.1.2 Ressources matérielles et virtuelles

La capacité et la performance d'un serveur sont définies par ses composants électroniques. Les principaux composants d'un serveur sont :

- le disque dur. Il stocke durablement les données à sauvegarder. Par exemple, le système d'exploitation et les applications installées sont stockés sur le disque dur. La capacité d'un disque dur est exprimée en gigaoctet (Go) voir en téraoctets (1 To = 1000 Go). La performance d'un disque dur se mesure par le temps de lecture/écriture des données. Pour un disque SATA, cette vitesse est au maximum de 300 Mo par seconde ;
- le processeur (aussi appelé microprocesseur ou CPU de l'anglais, Central Proccessing Unit)

est le composant qui exécute les calculs liés à l'exécution des applications et des instructions. La capacité d'un processeur correspond au nombre de calculs qu'il est capable d'exécuter en une seconde. Cette fréquence est exprimée en gigahertz (GHz). Afin d'exécuter plus efficacement des calculs en parallèle, un serveur peut posséder plusieurs processeurs ;
- la mémoire vive (RAM, Random Access Memory en anglais) stocke les données nécessaires aux calculs exécutés par le processeur. La mémoire vive se distingue du disque dur par une vitesse de lecture/écriture des données importantes (jusqu'à 17 Go par seconde pour de la RAM DDR3). Cependant, les données stockées en mémoire vive sont perdues à l'extinction du serveur. Il est donc impossible d'utiliser ce type de mémoire pour le stockage du système d'exploitation. La capacité d'une mémoire s'exprime en mégaoctets (Mo) ou en gigaoctects (1 Go = 1000 Mo) ;
- la carte réseau permet au serveur de communiquer avec d'autres serveurs ou appareils informatiques.

Le système d'exploitation est responsable de la distribution des ressources matérielles citées ci-dessus aux applications qu'il exécute. Avec l'ajout de la virtualisation, l'hyperviseur distribue les ressources matérielles aux VM et le système d'exploitation de chaque VM distribue les ressources virtuelles aux applications 2.1. Les ressources virtuelles d'une VM sont configurables par le biais des périphériques virtuels reliés à chaque VM. Ces périphériques reprennent les différents composants d'un serveur et sont associés à une VM avant son démarrage :

- le disque dur d'une VM est représenté par une image disque. Cette image est un fichier de taille suffisamment importante pour pouvoir stocker l'installation d'un système d'exploitation et de plusieurs applications. Différents formats d'image disque existent :
  - le format « raw » est un format dit « brut », c'est-à-dire sans optimisation et compression. C'est le format d'images le plus simple à manipuler,
  - le format « qcow » utilise la technologie « Copy On Write ». L'image disque a une taille correspondant à la quantité de disque réellement utilisée. Par exemple, une image disque de 8 Go maximum sur laquelle un système d'exploitation occupant 1 Go est installé aura une taille de 1 Go. La taille de l'image disque grandira alors à chaque installation d'applications jusqu'à atteindre la limite de 8 Go,
  - le format « qcow2 » permet de créer plusieurs images disque à partir d'une image disque principale accessible en lecture seulement. Les changements survenant dans chaque VM sont alors stockés dans des fichiers séparés. Cette technologie est pratique pour la création de VM possédant les mêmes applications ;
- la capacité CPU d'une VM dépend du nombre de processeurs virtuels (appelés VCPU, Virtual Central Processing Unit en anglais). Plus une VM possède de VCPU, plus l'hyperviseur lui attribuera une quantité importante de CPU. Cependant, un VCPU n'est pas lié à un CPU. Un serveur de deux CPU peut héberger deux VM possédant chacune quatre VCPU. L'hyperviseur ordonnance ensuite les VM suivant la quantité de CPU qu'elle demande. Le but est d'éviter à une VM de monopoliser toute la ressource CPU du serveur et de mettre en famine les autres VM. Pour se faire, l'ordonnanceur de l'hyperviseur utilise des algorithmes d'ordonnancement proches de ceux utilisés dans les systèmes d'exploitation pour l'ordonnancement de processus [QL11a] ;
- la mémoire vive utilisable par une VM est définie lors du lancement de celle-ci. Certains hyperviseurs proposent la fonctionnalité de mémoire partagée entre VM (Transparent Page Sharing en anglais) [Wal02]. Le but est d'utiliser qu'une seule fois l'espace mémoire commun entre plusieurs VM, par exemple, l'espace mémoire d'un même système d'exploitation. Trois VM utilisant 400 Mo de mémoire peuvent alors s'exécuter sur un serveur possédant 1000 Mo de mémoire. Afin d'affiner la gestion de la mémoire des VM, de plus en plus d'hyperviseurs

utilisent le « ballooning » [Wal02]. Cette fonctionnalité permet à l'hyperviseur de prendre une partie de la mémoire d'une VM pour la donner à une autre. Pour cela, un « ballon » situé à l'intérieur de la VM est gonflé afin de consommer de la mémoire. La mémoire consommée par le « ballon » ne peut alors être utilisée par la VM. L'hyperviseur va alors dégonfler le « ballon » d'une autre VM afin que cette dernière puisse obtenir d'avantage de mémoire ;

- plusieurs périphériques réseau peuvent être définies pour une VM afin de reproduire le comportement d'un serveur possédant plusieurs cartes réseau. Toutes les communications réseau des VM sont gérées par l'hyperviseur. Trois modes de configuration sont disponibles pour accéder à une VM :
  - le mode « Host Only », littéralement « l'hébergeur seulement ». Seul l'hyperviseur peut communiquer avec la VM. La VM n'a pas accès au réseau.
  - le mode NAT (de l'anglais Network Address Translation). La VM utilise la configuration réseau (c.-à-d., l'adresse IP) de l'hyperviseur pour communiquer. La VM a accès au réseau mais vu du réseau, la VM n'est pas visible.
  - le mode Bridged. La VM utilise sa propre configuration réseau. Sur le réseau, la VM et l'hyperviseur apparaisse comme deux entités indépendantes. La VM a alors un accès total au réseau.

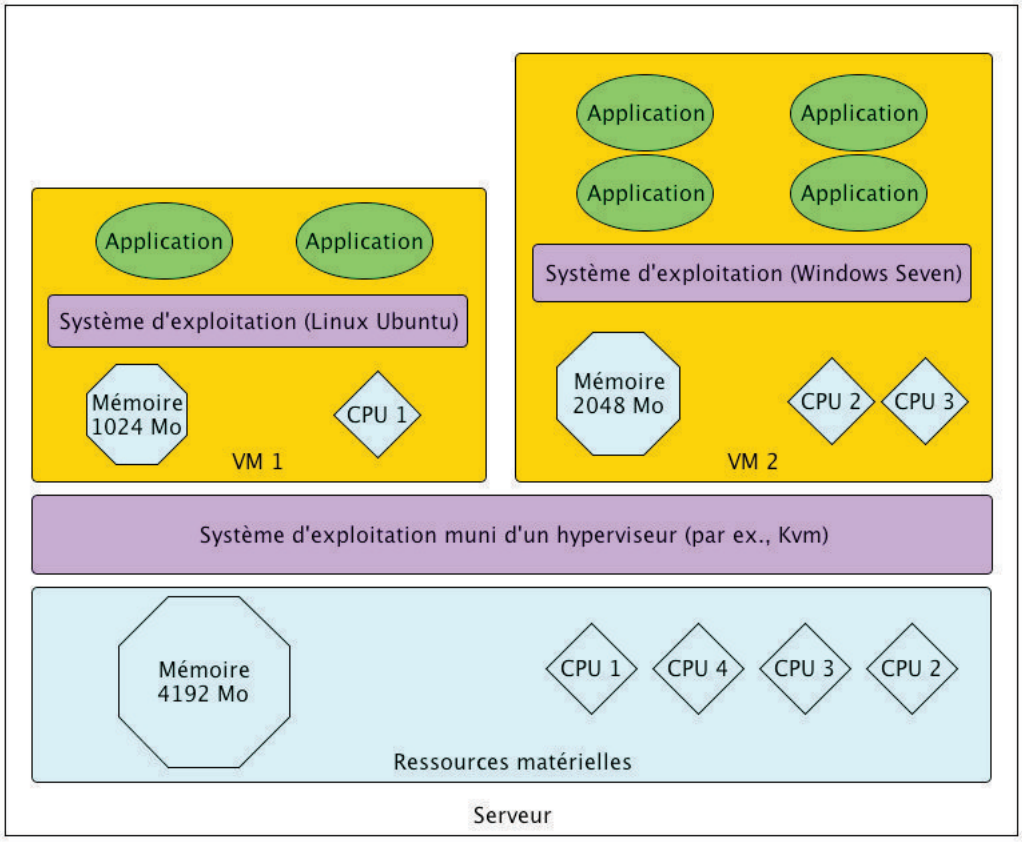

FIGURE 2.1 – Un serveur hébergeant des machines virtuelles

2.1.3 Migration de machines virtuelles

La migration consiste à déplacer une VM d'un hyperviseur à un autre afin de réorganiser la consommation de ressources. Une communication réseau entre les serveurs est nécessaire pour cette opération. Il existe deux types de migration : la migration à froid et la migration à chaud.

La migration à froid consiste à éteindre la VM, à transférer son image disque sur un autre hyperviseur en utilisant une communication réseau puis à allumer la VM une fois le transfert terminé. Les calculs en cours d'exécution ainsi que les données non sauvegardées (stockées en mémoire) sont définitivement perdus. De plus, la taille de l'image disque d'une VM peut être conséquente et donc son transfert peut être long. Afin de réduire le temps de la migration, il est très courant, pour ne pas dire systématique, que les hyperviseurs utilisent un disque dur externe accessible via le réseau. Le disque dur devient alors un stockage partagé par les hyperviseurs et donc, toutes les images disque sont elles aussi partagées. Le transfert de l'image disque n'est donc plus utile et la migration est effectuée très rapidement.

Grâce à l'usage très répandu des stockages partagés, une VM peut-être déplacée d'un serveur à un autre sans perdre les données présentes en mémoire. Dans ce cas, l'extinction de la VM est remplacée par une mise en pause. La mise en pause d'une VM consiste à écrire la mémoire de la VM dans l'espace de stockage. Cet espace étant partagé par plusieurs serveurs, la VM peut être allumé sur un autre serveur. Un fois la mémoire chargée, les calculs de VM reprennent. Cependant, ce type de migration est coûteux car l'écriture de la mémoire sur le disque dur du serveur nécessite du temps et un espace de stockage important (une VM peut posséder plusieurs gigaoctets de mémoire) est requis. De plus, la VM cesse de fonctionner entre sa mise en pause et son redémarrage sur un autre serveur. Ce comportement n'est pas acceptable pour les applications de haute disponibilité comme certains serveurs web.

La migration à chaud [CFH+05] pallie ce problème en permettant de déplacer une VM sans l'arrêter. Le transfert de l'image disque n'étant plus nécessaire, le réseau va être utilisé pour transférer la mémoire de la VM. Afin de réduire le temps d'arrêt de la VM, l'hyperviseur commence par transférer la mémoire non utilisée. Afin de finaliser la migration, les calculs de la VM sont suspendus pendant un court instant (quelques millisecondes) et le reste de la mémoire est copié. L'exécution de la VM reprend alors sans perte de données.

Lors de la migration d'une VM, la configuration réseau de celle-ci doit être actualisée. En effet, la VM, afin d'être accessible, possède une adresse IP unique dans son réseau. Si les serveurs source et destination ne sont pas dans le même réseau, l'adresse IP n'est plus reconnue après la migration et donc la VM n'est plus joignable. De plus, la configuration d'un réseau dans un centre de données de plusieurs centaines de serveurs peut être compliquée. Par exemple, les plateformes IaaS peuvent vouloir fournir un réseau à chaque client afin de garantir une isolation des réseaux et donc des communications plus sûres entre les VM d'un même client. Une telle granularité dans la configuration du réseau est difficilement accessible à partir d'équipements matériels comme les commutateurs (switch, en anglais). Des commutateurs virtuels (vSwitch, en anglais) sont donc apparus pour créer des réseaux dits virtuels permettant la mise en place de réseau entre VM.

2.1.4 Stockages partagés

Le partage de fichiers entre plusieurs serveurs ayant accès à un même réseau est possible par l'intermédiaire de systèmes de fichiers en réseau, aussi appelés systèmes de fichiers partagés. Le principe est de mettre à disposition de plusieurs serveurs le même espace de stockage. Un serveur possédant généralement une capacité de stockage importante va mettre à disposition de plusieurs clients son espace de stockage. Ce serveur est alors appelé « serveur de stockage ». Les clients utilisent alors l'espace de stockage par le biais de communications réseau. De nombreux

systèmes de fichiers partagés existent comme NFS (Network File System) créé pour les systèmes UNIX ou encore AFS (Andrew File System) inspiré de NFS. Ces systèmes de fichiers ont pour avantage d'être simples et rapides à mettre en place. Cependant, ils ne gèrent pas la tolérance aux pannes. Si un serveur de stockage est victime d'une panne, toutes ces données sont alors perdues. Afin de garantir la consistance des données même en cas d'incident sur les serveurs de stockage, les espaces de stockage de plusieurs serveurs sont agrégés en un seul. Dans cet espace de stockage, les données sont répliquées sur plusieurs serveurs. En cas de panne d'un serveur, les données ne sont donc pas perdues. Les systèmes de fichiers comme HadoopFS [SRC10] ou GoogleFS [GGL03] permettent de partager entre de nombreux clients un même espace de stockage de taille conséquente en étant tolérant aux pannes.

Afin d'améliorer les performances d'accès aux espaces de stockage, des matériels spécialisés comme les NAS (Network Area Storage) et les SAN (Storage Area Network) sont alors requis. Le NAS est un serveur spécialisé dans le partage de données. Il est accessible par ces clients via les systèmes de fichiers partagés précédemment cités. Un NAS possède un espace de stockage plus important que les serveurs traditionnels. Son stockage est aussi optimisé afin d'obtenir des accès rapides aux données. Si un seul NAS est utilisé par un trop grand nombre de clients, la qualité du réseau risque d'être dégradée en raison d'une quantité de données échangées trop importante. Dans ce cas, un SAN devient nécessaire afin d'utiliser un réseau propre à la gestion du stockage. Les multiples connexions réseau d'un SAN permettent aussi de multiplier les chemins d'accès aux espaces de stockage et donc d'éviter les problèmes liés aux pannes des équipements réseau.

À partir de ces espaces de stockage, les images disque des VM vont être accessibles par plusieurs hyperviseurs et donc faciliter la migration des VM. De plus, certains systèmes de fichiers partagés prennent en compte la tolérance aux pannes et donc limitent les pertes de VM contenant des données critiques.

2.1.5 Avantages et inconvénients

Exécuter plusieurs systèmes d'exploitation sur un serveur peut sembler étrange dans un premier temps. Cependant, dans les centres de données, de nombreux serveurs sont sous-utilisés. La virtualisation [FS08] permet le rassemblement de plusieurs serveurs consommant peu de ressources sur un seul. Il sera alors possible de diminuer le nombre de serveurs allumés et donc de réduire l'électricité consommée. Outre le fait de réduire le nombre de serveurs utilisés et les coûts associés, la virtualisation permet :

- de faciliter le déploiement d'applications ou de systèmes d'exploitation. Toute VM peut être clonée fournissant alors rapidement une nouvelle VM prêt à l'emploi ;
- d'isoler les applications sur un même serveur. Si une application est corrompue ou pose problème, on peut redémarrer la VM sans affecter les autres VM et leurs applications ;
- déplacer une application d'un serveur à un autre sans interrompre son fonctionnement grâce à la migration à chaud. Par exemple, déplacer une application sur un serveur plus puissant ;
- de sauvegarder facilement des VM.

Cependant, quelques inconvénients existent autour de la virtualisation :

- une dégradation des performances des applications s'exécutant dans les VM [HLAP06] est observée. Les applications consomment entre 5 et 10 % de ressources supplémentaires ;
- la hausse du coût d'administration des serveurs. Il faut ajouter l'administration des VM ;
- en cas de panne matérielle sur un serveur, toutes les VM de l'hyperviseur sont impactées ;
- la corruption d'une VM peut impacter la sécurité des autres VM hébergées ;
- le partage des ressources et des périphériques peut engendrer des pertes de performances ;

2.2 L'informatique dans les nuages : la virtualisation à l'échelle planétaire

2.2.1 Définition

Depuis le Minitel, l'utilisation d'un service s'exécutant sur un serveur distant est à la portée du grand public. Le serveur héberge des données et/ou des applications que le client consulte via un terminal léger (c.-à-d., un appareil possédant des ressources limitées). Avec Internet, les services accessibles à distance via un navigateurs web sont :

- nombreux. Par exemple, le nombre de sites web (et a fortiori de serveurs web) est en constante augmentation ;
- gourmands en ressources. Les services sont de plus en plus complexes et consomment donc plus de ressources (par ex., Google Maps et Facebook) ;
- très utilisés, notamment avec l'émergence des smartphones et l'arrivée de système d'exploitation comme Google Chrome OS, un système d'exploitation utilisant uniquement des applications distantes.

Afin de pouvoir répondre à des clients de plus en plus nombreux, un service web est composé de plusieurs applications hébergées par plusieurs serveurs localisés dans des centres de données différents. Du point de vue du client, un seul service est accessible. On parle alors d'informatique dans les nuages (Cloud computing, en anglais).

Les services proposés par l'informatique dans les nuages sont hébergés dans des centres de données et ont des besoins importants en termes de flexibilité (paramétrage du nombre d'applications composant un service, migration d'une application d'un centre de données à un autre, sauvegarde de données). Cette flexibilité est apportée par l'utilisation de la virtualisation.

2.2.2 Les grilles de calcul

L'offre de services à partir d'une infrastructure informatique n'est pas un besoin récent en informatique. Les travaux autour des grilles de calculs [TWML01, FK96] sont anciens et tentent de mettre à disposition d'un groupe d'utilisateurs des ressources informatiques conséquentes. Les concepts partagés autour de ces travaux permettent de définir une grille de calculs comme une infrastructure :

- partagée : l'infrastructure est mise à disposition de plusieurs utilisateurs ;
- distribuée : les serveurs composant l'infrastructure sont situés dans des lieux géographiquement différents ;
- hétérogène : les serveurs ont des configurations matérielles différentes et les applications hébergées sont diverses et variées ;
- coordonnée : l'exécution des tâches sur la grille est ordonnancée afin de garantir le bon fonctionnement des applications hébergées.

Cependant, les grilles de calculs visent, bien souvent, un public précis : le monde scientifique. Par exemple, Datagrid [PGW03], rassemblant jusqu'à 15000 serveurs répartis sur 125 sites en Europe, est mise à disposition de 500 scientifiques travaillant dans trois domaines nécessitant une forte puissance de calculs : la physique des hautes énergies, les applications biomédicales et l'observation de la terre. Le besoin de mettre un prix sur l'utilisation des ressources délivrées [CA07] par les grilles mène à une transition vers l'informatique dans les nuages.

L'informatique dans les nuages poursuit les travaux sur les grilles en mettant à disposition de tout utilisateur des ressources dites « illimitées ». Les fournisseurs d'infrastructures de l'informatique dans les nuages se voient donc obligés d'augmenter le nombre de leurs serveurs régulièrement

afin de suivre la demande et de ne pas manquer de ressources. La quantité de ressources disponibles permet aux utilisateurs d'avoir un accès instantané aux ressources qu'ils demandent. De plus, un effort particulier est porté sur l'élasticité apportée par de telles infrastructures permettant à l'utilisateur de facilement ajuster les ressources qu'ils consomment. La frontière entre les grilles et l'informatique dans les nuages a encore diminué lorsque la virtualisation est apparue dans les grilles afin de faciliter la gestion des ressources [FDF03]. Un particulier ou une entreprise peut désormais obtenir une ressource informatique sans acheter des serveurs et un lieu pour les stocker, sans avoir de connaissances spécifiques dans l'administration de serveurs, sans avoir à installer et configurer un système d'exploitation et ses applications.

2.2.3 Les différentes offres de l'informatique dans les nuages

L'informatique dans les nuages (Cloud computing, en anglais) proposent d'utiliser l'informatique comme un service s'apparentant à l'eau ou à l'électricité afin de payer seulement les ressources consommées. L'utilisateur ne loue plus des serveurs et leurs équipements mais paie en fonction des ressources qu'il utilise. Les ressources à louer sont variées : emplacements pour des serveurs dans un centre de données, serveurs, machines virtuelles, applications. Le fournisseur doit alors s'adapter à la demande en offrant des ressources informatiques variées (voir figure 2.2) :

- « Infrastructure as a Service » (IaaS) : le client obtient des VM qu'il doit administrer ;
- « Platform as a Service » (PaaS) : le client développe ses applications avec les langages de programmation mis à sa disposition (par ex., .NET pour la plateforme Microsoft Azure). L'application est ensuite automatiquement déployée dans le centre de données ;
- « Software as a Service » (SaaS) : le client utilise un service fourni par une application déjà déployée dans un centre de données (par ex., un outil de messagerie en ligne comme Microsoft Hotmail) ;
- « Desktop as a Service » (DaaS) : le poste de travail est hébergé dans le centre de données. Le client accède à son poste via une connexion Internet ;
- « Data as a Service » (DaaS) : le client a accès à des données et des outils pour les traiter et les visualiser (par ex., Google Data Explorer).

Les différents services de l'informatique dans les nuages sont offerts par des centres de données [CGH09] :

- externes à l'entreprise. L'entreprise paie généralement le service au fournisseur qui gère le centre de données. On parle alors de « cloud public ». Amazon ou Microsoft Azur en font parties ;
- internes à une ou plusieurs entreprises. La gestion du centre de données est à la charge des entreprises. On utilise le terme de « cloud privé » ;
- externes et internes. On parle alors de « cloud hybride ». Ce système est adopté afin d'agrandir un « cloud privé », par exemple, lors de pics de charge.

L'utilisation d'un cloud public permet à une entreprise d'obtenir des ressources informatiques sans avoir à investir dans un centre de données. L'entreprise ne possède donc ni le matériel (serveurs, climatisation, etc.), ni les administrateurs gérant les ressources informatiques qu'elle consomme. L'administration est donc entièrement déléguée à un tiers qui assure une certaine confidentialité et des garanties de sécurité. Cependant, certaines entreprises nécessitent une administration particulière (par ex., une politique d'équilibrage de charge) et/ou une confidentialité accrue des données qu'elle utilise. Dans ces cas, un cloud privé est souvent préféré afin de mettre en place une administration personnalisée et efficace. Le cloud hybride permet un mixe des deux technologies décrites précédemment. Un cloud hybride est souvent utilisé afin d'augmenter la taille d'un cloud privé soit continuellement pour héberger des services de moindre importance

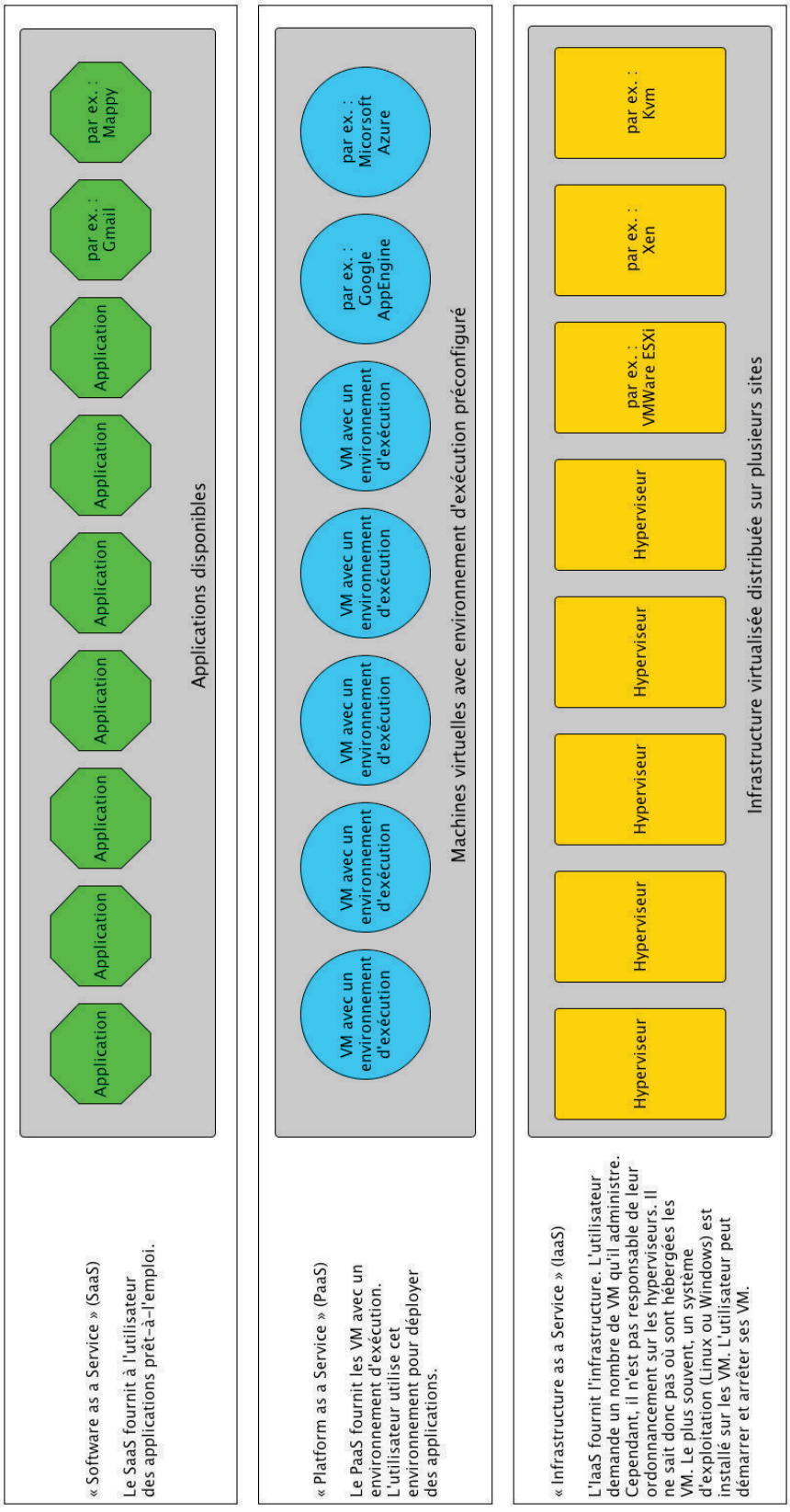

FIGURE 2.2 – Les différents niveaux de l'informatique dans les nuages

soit ponctuellement pour répondre à un pic de charge nécessitant des ressources supplémentaires.

2.2.4 Avantages et inconvénients

Les avantages de l'informatique dans les nuages sont :
- les avantages liés à l'utilisation des centres de données (centralisation d'équipements de qualité, sécurité...) ;
- d'éviter la multiplication des applications et du personnel nécessaire au bon fonctionnement des services ;
- la gestion centralisée de nombreux services permet la mise en place de politique de rationalisation, par exemple, une politique d'économie d'énergie.

Les principaux inconvénients de l'informatique dans les nuages sont liés à l'opacité engendrée par ce service. D'un point de vue de la sécurité, les utilisateurs ne savent pas où et comment sont stockées leurs données. Pour une entreprise, certaines données sont critiques et ne doivent pas être divulguées. De plus, peu de garanties sont proposées en termes de qualité de service. L'entreprise a peu de moyens d'action en cas de services dégradés ou interrompus.

2.3 Les centres de données au service de l'informatique dans les nuages

Les infrastructures servant l'informatique dans les nuages sont les centres de données. En effet, la conception de bâtiments dédiés à l'hébergement de nombreux serveurs permettent d'améliorer l'exploitation des ressources informatiques en utilisant du matériel de qualité. Par exemple, à l'intérieur d'un centre de données, les communications entre serveurs sont optimisées afin de réduire le temps des migrations de VM. Outre l'apport d'équipements spécialisés, le centre de données apporte une gestion globale d'un grand nombre de VM et donc d'applications. Des outils sont alors nécessaires afin d'obtenir des informations sur l'infrastructure et d'organiser les ressources consommées.

2.3.1 Une structure complexe

L'administrateur a pour rôle d'organiser les ressources de l'infrastructure. Ces ressources sont de deux types : les ressources consommées et les ressources fournies. La complexité de l'infrastructure virtualisée vient du fait qu'un élément peut consommer des ressources et en fournir d'autres. Par exemple, un serveur fournit de la mémoire et de la ressource CPU mais consomme de l'électricité et de l'air froid (climatisation). L'ajout d'une VM sera faite en analysant les capacités mémoire des serveurs de l'infrastructure tandis que l'ajout d'un serveur dans un centre de données devra être fait en connaissant la topologie électrique de l'infrastructure.

En effet, les alimentations des serveurs sont doublées afin d'éviter l'arrêt du serveur en cas de défaillance d'une de ces alimentations. Dans le but d'ajouter un niveau de sécurité, chaque alimentation est branchée sur un générateur d'électricité différent. En cas de panne d'un générateur, le deuxième générateur prend le relais et la panne n'impacte donc pas les serveurs. Cependant, chaque générateur ne peut fournir qu'une quantité limitée d'électricité. Si un générateur tombe en panne et que le second générateur ne peut pas alimenter la totalité des serveurs, alors un risque de panne sur le second générateur existe. De plus, les serveurs ne sont pas les seuls éléments électriques alimentés par les générateurs. La climatisation ou encore les équipements réseau sont aussi branchés sur les générateurs. Il est donc important de pouvoir décrire

l'organisation électrique d'un centre de données afin d'organiser les serveurs de manière à limiter l'impact d'une panne de générateur.

La description de l'organisation des serveurs doit aussi servir à localiser facilement un serveur dans le centre de données. Dans un centre de données, les serveurs sont rangés dans des armoires (rack, en anglais) pouvant contenir plusieurs dizaines de serveurs. En cas de panne d'une alimentation, la description permet de retrouver l'armoire stockant le serveur. Il est alors aisé de retrouver le serveur a réparé.

Ce besoin de description présenté pour les schémas électrique et physique est un besoin générique de description et peut-être complété ou remplacé par d'autres vues : thermique, climatisation, sécurité, etc. Une des vues est utilisée pour décrire l'organisation des VM. Cette vue doit être renseignée afin de savoir sur quels serveurs une VM peut démarrer. Par exemple, dans le cas de stockages partagés, l'image disque d'une VM servant à son démarrage est accessible par plusieurs serveurs. Le renseignement des stockages partagés est aussi utile pour effectuer une migration à chaud. L'administrateur doit être capable de lister les serveurs ayant accès à l'image disque de la VM et donc éligibles à la migration. La description de l'organisation des VM est aussi essentielle pour la manipulation de celles-ci. Par exemple, l'association d'une VM à un projet ou à un utilisateur permet d'effectuer des reconfigurations facilement et avec plus de sécurité. Par exemple, l'extinction des VM de l'utilisateur « lambda » ou des VM du projet « GreenIt » peut devenir périlleuse si la description fournie est incomplète. On risque de laisser des VM non utilisées s'exécuter et d'éteindre des VM appartenant à un autre projet ou à un autre utilisateur.

La connaissance de l'organisation électrique des serveurs peut aussi influencer l'organisation des VM sur l'infrastructure. Suivant la taille de l'infrastructure, l'âge de ces serveurs et la politique de l'entreprise, tous les serveurs ne sont pas forcément relié à un générateur de secours. Par exemple, seuls les serveurs les plus récents (et donc les plus puissants) peuvent être relié à ce générateur. En cas de panne, les serveurs fournissant le plus de ressource sont maintenus en état de fonctionnement. Cependant, garantir le fonctionnement des serveurs les plus sûrs est inutile s'ils n'hébergent pas les VM fournissant les services les plus critiques. Pour mettre en place et surtout maintenir une telle organisation, une connaissance précise des VM et de l'organisation électrique des serveurs est indispensable.

La description d'une infrastructure virtualisée doit donc être la plus complète possible afin de permettre ou de faciliter les tâches d'administration. Une certaine flexibilité est aussi nécessaire afin de pouvoir prendre en compte les différentes topologies existantes. Par exemple, une infrastructure peut être composée de plusieurs centres de données localisés dans des villes différentes. L'association entre une ville et un serveur permet alors d'éviter de migrer une VM d'une ville à une autre et donc d'effectuer des migrations utilisant des communications réseau moins performantes.

2.3.2 Observation de l'infrastructure

La description de l'infrastructure donne une vision statique des ressources. Le nombre de centres de données présents dans une infrastructure ou la quantité de CPU et de mémoire fournie par un serveur varient rarement. Il en est de même pour la configuration des stockages partagés qui sont rarement modifiés pour simplifier l'administration.

Cependant, de nombreuses ressources varient au fil du temps. Des VM vont être ajoutées ou retirées ce qui va, respectivement, diminuer ou augmenter les ressources consommées par un serveur. De plus, la consommation d'une VM varie en fonction de son utilisation. Par exemple, les consommations CPU et réseau d'un serveur web dépendent du nombre de clients connectés

au serveur. Plus il y a de clients, plus le serveur consomme.

Afin de connaître la quantité de ressources consommées par les serveurs et les VM, des outils d'observation sont nécessaires. Ces outils permettent d'obtenir la quantité de ressources consommées à un instant donné. En enrichissant la description de l'infrastructure avec les données obtenues à partir des outils d'observation, une vue globale de l'organisation des ressources peut être construite. L'administrateur peut alors connaître les ressources consommées par un serveur afin, par exemple, de lui ajouter une VM. Si trop peu de ressources sont disponibles, un autre serveur doit être trouvé.

Ces outils d'observation sont principalement destinés à récupérer les ressources consommées sur les serveurs et les VM. Cependant, d'autres outils peuvent être installés afin de récupérer la consommation d'autres éléments comme la récupération de l'électricité consommée par un climatiseur. Il existe deux façons d'obtenir les ressources consommées pour une VM ou pour un serveur. La première consiste à installer un outil d'observation à l'intérieur de la VM ou du serveur. La seconde consiste à récupérer les ressources consommées à partir de communications réseau avec le serveur. Un outil peut alors collecter les informations de plusieurs serveurs. Sur chaque serveur, l'hyperviseur est capable de calculer les ressources consommées des VM qu'il héberge. Les valeurs mesurées sont alors moins précises notamment lorsque très peu de ressources sont disponibles sur le serveur. Cependant, cette méthode est privilégiée dans les infrastructures virtualisées car elle évite l'installation et la configuration d'un outil d'observation sur chaque serveur et sur chaque VM. La configuration d'un tel système d'observation sur des infrastructures de plusieurs centaines de serveurs entraînent une hausse importante des tâches d'administration.

2.3.3 Introspection et reconfiguration de l'infrastructure

La description de l'infrastructure et les outils d'observation mettent à disposition de l'administrateur les données statiques et dynamiques nécessaires à une organisation correcte des ressources de l'infrastructure. Cependant, dès que la taille de l'infrastructure dépasse la dizaine de serveurs, la quantité de données relatives à l'infrastructure devient importante. Un outil est alors nécessaire afin d'agréger les données statiques et dynamiques et d'offrir à l'administrateur une interface permettant la récupération de ces données. Par exemple, l'administrateur peut consulter la consommation CPU de toutes les VM d'un projet donné.

À partir de ces informations, l'administrateur organise les ressources de l'infrastructure afin d'éviter divers problèmes comme une VM n'obtenant pas suffisamment de ressources pour fonctionner correctement. La réorganisation de l'infrastructure dans le but d'ajouter, de supprimer ou de répartir des VM est faite à partir d'opérations appelées reconfigurations. Ces reconfigurations utilisent la flexibilité apportée par la virtualisation pour, respectivement, démarrer, arrêter ou migrer des VM.

L'administrateur peut aussi organiser les ressources dans le but de minimiser la consommation électrique de l'infrastructure ou d'équilibrer la consommation de ressources sur chaque serveur. Dans le premier cas, l'administrateur cherchera à regrouper toutes les VM sur un minimum de serveurs tout en garantissant les demandes en ressource des VM en cours d'exécution. Les serveurs non utilisés, c'est-à-dire n'hébergeant pas de VM, seront alors éteints pour diminuer la consommation énergétique de l'infrastructure. On parle de « consolidation ». Dans le second cas, l'administrateur cherchera à équilibrer la consommation en ressource de chaque serveur afin de conserver un maximum de ressources disponibles sur chaque serveur. Si une VM demande plus de ressources, les ressources disponibles seront utilisées. On parle « d'équilibrage de charge ».

Sur un serveur sans hyperviseur, les applications sont réparties sur les processeurs du serveur suivant une « politique d'ordonnancement ». Aux vues des nombreux points communs existants

entre les VM et les applications [QL11a], le terme « politique d'ordonnancement » est repris pour définir l'organisation des VM sur les serveurs de l'infrastructure.

2.3.4 Vers des systèmes plus sûrs

La politique d'ordonnancement définit l'organisation globale des VM dans l'infrastructure. Cependant, certains besoins précis liés à des problématiques variées doivent pouvoir être spécifiés par l'administrateur. Par exemple, l'administrateur doit pouvoir regrouper des VM communicant entre elles sur un même serveur afin d'optimiser leur fonctionnement. Pour répondre à des objectifs de tolérance aux pannes, l'administrateur doit être capable d'éviter que des VM rendant le même service soient hébergées sur un seul serveur. Si le serveur connaît une défaillance, le service devient indisponible. Ces VM ne doivent donc pas cohabiter. La spécification de ces besoins sera faite à l'aide de « règles de placement ».

Pour une infrastructure de petite taille (par ex., une dizaine de serveurs), l'administrateur peut vérifier les règles de placement à l'aide des données retournées par l'outil d'observation. La vérification se corse avec une infrastructure d'une centaine de serveurs. Des outils sont alors indispensables pour analyser les données d'observation et remonter à l'administrateur les éventuels problèmes détectés. Ces problèmes peuvent être des règles de placement non respectées ou des serveurs dans l'incapacité de répondre aux besoins des VM qu'il héberge. En effet, l'évolution des ressources consommées par les VM peut entraîner une consommation totale des ressources du serveur. On dit alors que le serveur est surchargé.

Afin de résoudre ces problèmes, l'administrateur calcule les reconfigurations à exécuter. Ce calcul peut être simple si peu de problèmes sont détectés mais il se complexifie rapidement si plusieurs serveurs sont surchargés et que de nombreuses règles de placement existent. Si l'administrateur doit aussi prendre en compte une politique d'ordonnancement, le problème à résoudre devient un réel casse-tête. Afin d'assurer des reconfigurations cohérentes, l'administrateur doit pouvoir être assisté par des algorithmes permettant le calcul des reconfigurations à appliquer afin de résoudre les problèmes en respectant la politique d'ordonnancement mise en place. Ces algorithmes sont appelés « algorithmes d'ordonnancement ».

2.4 Conclusion

Avec la démocratisation de l'informatique dans les nuages, l'utilisation des centres de données est de plus en plus courante pour regrouper les ressources informatiques. L'augmentation des services web et de l'Internet mobile à entraîner une hausse de la taille des centres de données ainsi qu'une complexification de leur organisation. Pour obtenir plus de flexibilité dans l'administration de ces infrastructures, la virtualisation est souvent plébiscitée. Cependant, l'utilisation de cette technologie augmente le nombre des tâches d'administration. Des outils sont alors nécessaires pour aider et assister les administrateurs à organiser les ressources de leur infrastructure. L'objectif du prochain chapitre est de proposer un état de l'art de ces outils.

Chapitre 3

L'administration des infrastructures virtualisées

L'administration d'une infrastructure virtualisée désigne l'ensemble des tâches nécessaires au bon fonctionnement des applications hébergées. L'évolution des besoins des applications dans une infrastructure de taille conséquente nécessite des outils adaptés pour aider l'administrateur à décrire, sélectionner et reconfigurer son infrastructure. Des gestionnaires de VM sont utilisés par l'administrateur pour visualiser la consommation de ressources et exécuter des reconfigurations via une interface graphique. Cependant, les interfaces graphiques sont mal adaptées à la manipulation de nombreux éléments et empêche l'utilisation de scripts. Afin d'aider l'administrateur à reconfigurer son infrastructure, des systèmes autonomes sont déployés sur l'infrastructure. Ces outils analysent périodiquement la consommation de ressources et les réorganisent en fonction d'une politique d'ordonnancement prédéfinie. L'administrateur n'est alors plus maître des reconfigurations exécutées et l'observation des ressources de l'infrastructure n'est plus accessible. Les langages dédiés représentent des outils capables de manipuler des ensembles importants d'éléments. Des langages de description d'infrastructures sont utilisés pour modéliser les infrastructures. Les langages de sélection se focalisent sur les ressources réseau, CPU, mémoire, etc. des éléments sans permettre la récupération de données organisationnelles. Les langages de reconfigurations manipulent les serveurs et les VM afin de réorganiser les ressources de l'infrastructure. Les éléments sont traités un à un ce qui complique l'administration d'une infrastructure contenant des centaines d'éléments nécessitant des manipulations ensemblistes.

Sommaire

3.1 L'administration et la virtualisation

Le terme « administration » décrit l'ensemble des tâches nécessaires au bon fonctionnement d'un centre de données [SG10] que ce soit au niveau de sa sécurité (par ex., la gestion du réseau), de son fonctionnement (par ex., la résolution des pannes), de son exploitation (par ex., la gestion des comptes des utilisateurs) ou de son évolution (par ex., la mise à jour des applications). L'administration s'intéresse donc aux trois niveaux de l'infrastructure : le niveau physique, le niveau logique et le niveau applicatif.

L'administration du niveau physique comprend la gestion du matériel au sens large. En effet, de nombreux équipements sont nécessaires au fonctionnement des serveurs. Des équipements de climatisation maintiennent une température autour de vingt degrés afin d'éviter la surchauffe des composants informatiques. Des équipements réseau comprenant des systèmes de détection d'intrusion logicielle, des pare-feux, des routeurs, des commutateurs rendent possible la communication entre les serveurs. De nombreux câbles servant pour les communications réseau ou pour l'alimentation électrique des différents appareils. Des caméras pour lutter contre les intrusions physiques et de nombreux capteurs pour mesurer la température, l'humidité, la quantité de poussière, etc. Tous ces équipements doivent être régulièrement révisés afin de garantir le fonctionnement optimal des serveurs. L'administration de ces derniers se résume à la configuration matérielle. Cette configuration définit les ressources matérielles du serveur comme la mémoire et l'espace de stockage. L'installation du système d'exploitation est régulièrement inclut dans la configuration matérielle. Lors de cette phase, un hyperviseur peut être installé afin de rendre possible l'utilisation de la virtualisation. Le fournisseur d'un cloud de type « Infrastructure As A Service » est seulement responsable de l'administration des serveurs de l'infrastructure. Les autres niveaux sont à la charge de l'utilisateur.

La flexibilité apportée par la virtualisation permet l'évolution du nombre de VM hébergées sur l'infrastructure en fonction des besoins des utilisateurs. L'administration du niveau logique comprend donc la gestion de la virtualisation. L'administration d'une VM se décompose en deux phases : la phase de configuration matérielle et la phase de reconfiguration. La phase de configuration matérielle définit les ressources utilisées par la VM comme la taille de sa mémoire et de son image disque. Comme pour les serveurs, cette phase inclut souvent l'installation d'un système d'exploitation. La phase de reconfiguration assure le bon fonctionnement des VM. Cette phase nécessite l'observation des ressources du serveur consommées par les VM hébergées Les ressources fournies par le serveur sont statiques, c'est-à-dire qu'elles n'évoluent pas dans le temps. Par conséquent, si les ressources nécessaires aux VM augmentent, les ressources du serveur peuvent devenir insuffisantes. Dans ce cas, la phase de reconfiguration réorganise les ressources consommées par les VM dans le but d'assurer aux VM l'obtention des ressources nécessaires à leur fonctionnement. Les modifications apportées lors de la phase de reconfiguration sont décrites par le cycle de vie de la VM détaillé par la figure 3.1. Par exemple, une VM peut être démarrée à l'aide de la reconfiguration « Start ». Dans l'état « Running », les applications mises à disposition par la VM sont accessibles. Par conséquent, la VM consomme les ressources offertes par son serveur hôte. Si l'application n'est plus utilisée ou si le serveur a besoin de récupérer des ressources, la VM est éteinte à l'aide de la reconfiguration « Stop ». Dans l'état « Off », la VM ne consomme plus de ressource CPU, mémoire et réseau.

La flexibilité apportée par la virtualisation est importante car elle permet à l'administrateur d'adapter l'organisation des VM de son infrastructure en fonction des consommations des applications. Par exemple, les sites de commerce comme Amazon sont plus utilisés en dehors des horaires de travail. À l'inverse, pour les services de « Desktop as a Service » hébergés par une infrastructure, les pics d'activité auront lieu lors des horaires de travail. L'administrateur

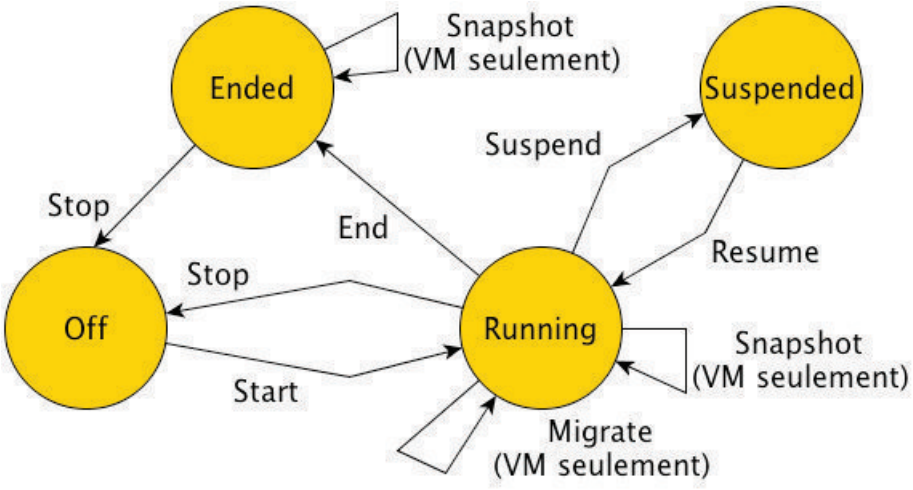

FIGURE 3.1 – Le cycle de vie d'un serveur ou d'une machine virtuelle. Les flèches montrent les reconfigurations modifiant l'état de la VM ou du serveur. L'état « ended » permet de spécifier que le serveur ou la VM doit être arrêté. Pour un serveur, cet état permet de programmer une maintenance. Le serveur doit être vidé de ses VM et ne peut pas accueillir de VM supplémentaires. Pour une VM, l'état « ended » permet à l'administrateur d'effectuer une sauvegarde avant l'extinction de la VM.

devra alors réorganiser le centre de données afin de prendre en compte ces variations d'activité. La flexibilité apportée par la virtualisation sera alors utilisée, par exemple, pour résoudre un problème de surcharge :

- en suspendant ou en arrêtant une VM afin de libérer plus de ressources pour les VM voisines (c.-à-d., les VM hébergées par le même serveur). Dans le cas d'une suspension, la VM sera réactivée et les applications reprendront leur exécution ;
- en migrant une VM vers un serveur ayant des ressources disponibles. Dans ce cas, l'exécution des applications n'est pas perturbée.

Si une VM manque de ressources, les applications qu'elle exécute peuvent connaître des dysfonctionnements allant jusqu'à l'arrêt de l'application. Lorsqu'un serveur ne peut plus fournir assez de ressources à toutes ses VM, le serveur est dit surchargé. En cas de surcharge importante, le serveur peut lui aussi s'arrêter. L'utilisation de la virtualisation augmente donc la flexibilité de l'infrastructure mais, en contrepartie, son administration se complexifie [KK09] car l'administration des VM est ajoutée à celle des serveurs.

L'administration des serveurs et des VM sont pris en charge par les fournisseurs d'un cloud de type « Platform As A Service ». La plupart des offres de « PaaS » proposent à l'utilisateur différents modèles de VM. Ces modèles se différencient par la quantité de ressources matérielles affectées à la VM et le système d'exploitation s'exécutant sur la VM. Une fois les VM choisies et déployées sur l'infrastructure du fournisseur, l'utilisateur les utilise pour déployer des applications. Les « PaaS » sont, par exemple, utilisés par les développeurs pour concevoir et/ou tester des applications destinées à une infrastructure de l'informatique dans les nuages.

L'administration du niveau applicatif comprend l'installation et la configuration des applications. Suivant le type de l'infrastructure, virtualisée ou non virtualisée, les applications sont installées, respectivement, sur des VM ou sur des serveurs. Les fournisseurs de cloud de type « Software As A Service » proposent à l'utilisateur une liste d'applications (ou de services) prêtes à l'emploi. Les utilisateurs n'ont donc plus aucune administration à effectuer.

L'étude proposée dans la suite de ce chapitre s'intéresse à l'administration des niveaux physique et logique. Un intérêt particulier est porté sur l'administration logique responsable de la gestion des VM hébergées par les serveurs. La taille des infrastructures virtualisées étant en constante augmentation, il devient difficile à un administrateur d'effectuer une gestion efficace de ses serveurs sans utiliser d'outils d'administration. Ces outils simplifient la configuration matérielle des serveurs et des VM, assurent l'observation des ressources consommées et facilitent la gestion du cycle de vie des serveurs et des VM.

3.2 Les outils d'administration dédiés aux infrastructures virtualisées

La consommation de ressources d'une VM est dynamique, c'est-à-dire, qu'elle évolue au fil du temps. Par exemple, la consommation CPU d'une application web dépend du nombre de ses utilisateurs. L'administrateur a donc besoin de reconfigurer son infrastructure afin de prendre en compte l'évolution des besoins des applications et donc des VM.

Grâce à la virtualisation, les infrastructures sont plus flexibles (création et suppression de VM) et facilement reconfigurables, notamment avec la migration de VM. De nombreux gestionnaires de VM existent. Ils peuvent être classés en deux groupes : les gestionnaires de VM (VMM, Virtual Machine Manager en anglais) et les ordonnanceurs autonomes de VM (VMS, Virtual Machine Scheduler en anglais).

3.2.1 Les gestionnaires de VM

Les VMM prennent en charge les différentes reconfigurations intervenant dans le cycle de vie des VM (voir figure 3.1). La première étape de ce cycle de vie est la création de l'image disque de la VM sur laquelle le système d'exploitation et les applications seront installés. La création de l'image disque terminée, les différentes ressources utilisées par la VM (mémoire, réseau, VCPU) sont configurées. L'installation d'un système d'exploitation est une opération coûteuse en temps. Afin de pouvoir créer des VM rapidement, de nombreux VMM proposent la fonctionnalité de clonage permettant de créer une VM à partir d'une VM existante. Dans ce cas, la nouvelle VM possède le même système d'exploitation et utilise la même quantité de ressources que la VM source. La VM est maintenant prête à être démarrée, elle est dans l'état « Off ». On dit que la VM est « instanciée » (sa phase d'instanciation est terminée). Une fois la VM démarrée, les VMM vont permettre de la migrer vers un autre serveur, d'en effectuer une sauvegarde, de l'arrêter, de la mettre en pause ou en maintenance grâce à l'état « Ended ». La mise en maintenance d'une VM est la dernière étape avant l'arrêt de celle-ci.

Dans une infrastructure de taille conséquente, une vision globale de l'infrastructure et des ressources consommées est nécessaire, par exemple, pour trouver les ressources nécessaires à la création d'une nouvelle VM. Pour pallier ce problème, les VMM proposent une gestion centralisée de la totalité de l'infrastructure via une interface graphique. Le nombre de VMM disponibles étant très important, la liste des travaux présentés dans le reste de cette section n'est pas exhaustive.

Oscar-V [VNS07] est un VMM spécialisé dans le déploiement de VM sur une grappe d'hyperviseurs Xen. À l'aide de fichiers XML, l'administrateur configure les applications à installer sur le système d'exploitation des VM à créer. L'outil V2M offre un shell proposant les reconfigurations liées au cycle de vie de la VM.

Eucalyptus [NWG+09] met en place une infrastructure de type « Infrastructure As A Service » tout en fournissant un outil de recherche modulaire. L'architecture déployée (voir figure 3.2) permet la gestion des VM, du stockage des images disque et des réseaux virtuels. Le « Cloud

Controller » offre des services compatibles avec ceux offerts par Amazon [Ama] et une interface web pour observer l'infrastructure déployée. Les « Cluster Controller » mettent en place la gestion du réseau et les politiques d'ordonnancement de VM. Pour finir, les « Node Controller », installés sur chaque hyperviseur, sont responsables de l'exécution des opérations de reconfiguration (par ex., la création et l'arrêt des VM).

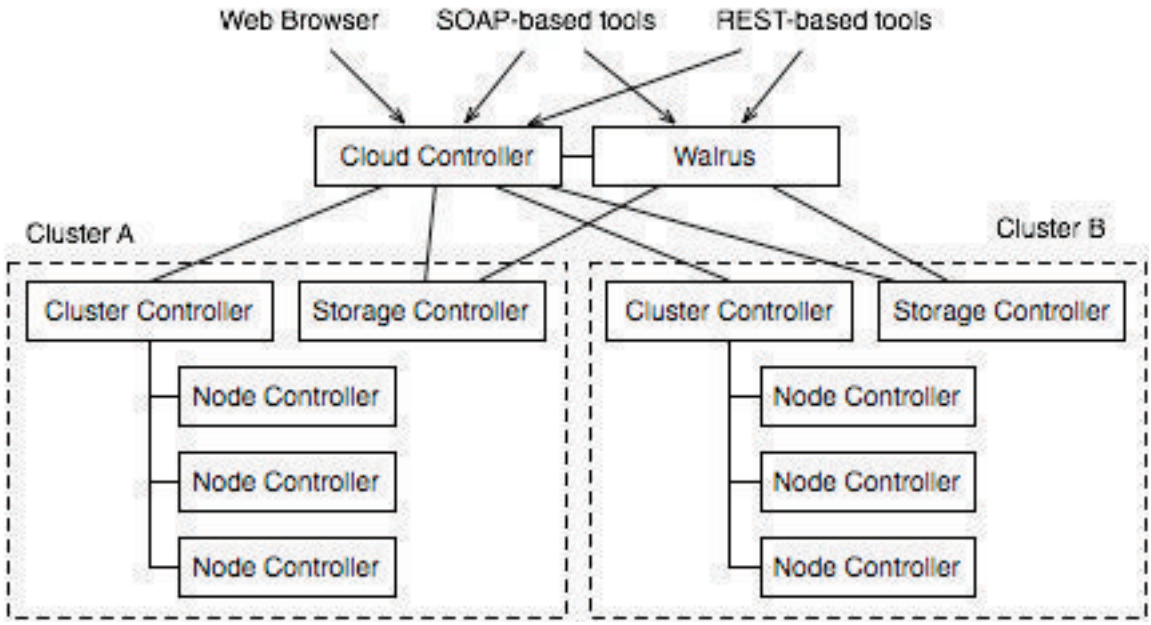

FIGURE 3.2 – Architecture de l'outil Eucalyptus

OpenNebula [SMLF09] est un gestionnaire de VM fournissant tous les outils nécessaires (voir figure 3.3) à l'administration de VM. Son architecture modulaire lui permet de s'adapter à de nombreuses infrastructures et de prendre en charge des fournisseurs d'IaaS externes comme Amazon.

VMware vSphere [VMwb] est le gestionnaire de VM de la firme VMware. L'outil propose une offre complète de virtualisation basée sur l'hyperviseur ESX. Plusieurs vues sont disponibles dans l'interface suivant les préoccupations de l'administrateur : stockage des images disque, définition de réseaux virtuels, affectation de VM à des hyperviseurs.

Virt-Manager [vir] est un VMM avec interface graphique Open Source. Il utilise libvirt pour gérer les hyperviseurs Xen et KVM et permet la visualisation de l'organisation des ressources de l'infrastructure.

Les VMM avec interface graphique offrent une vue unifiée de l'infrastructure et permettent à l'administrateur d'effectuer des opérations de reconfiguration en ayant connaissance de la répartition des ressources. Cependant, une interface graphique n'est pas adaptée à la manipulation d'ensembles de VM et de serveurs de taille importante. Par exemple, l'arrêt d'une centaine de VM impose, dans le meilleur des cas, cent sélections de VM pour ensuite ordonner l'arrêt. De plus, l'emploi d'une interface graphique empêche l'utilisation de scripts permettant de décrire, de sauvegarder et d'exécuter de multiples fois une séquence de reconfigurations. Les gestionnaires de VM aident l'administrateur à configurer et reconfigurer son infrastructure à l'aide des reconfigurations qu'il propose. Cependant, en cas de problème, aucune solution n'est proposée à l'administrateur qui doit alors calculer les reconfigurations nécessaires par lui-même.

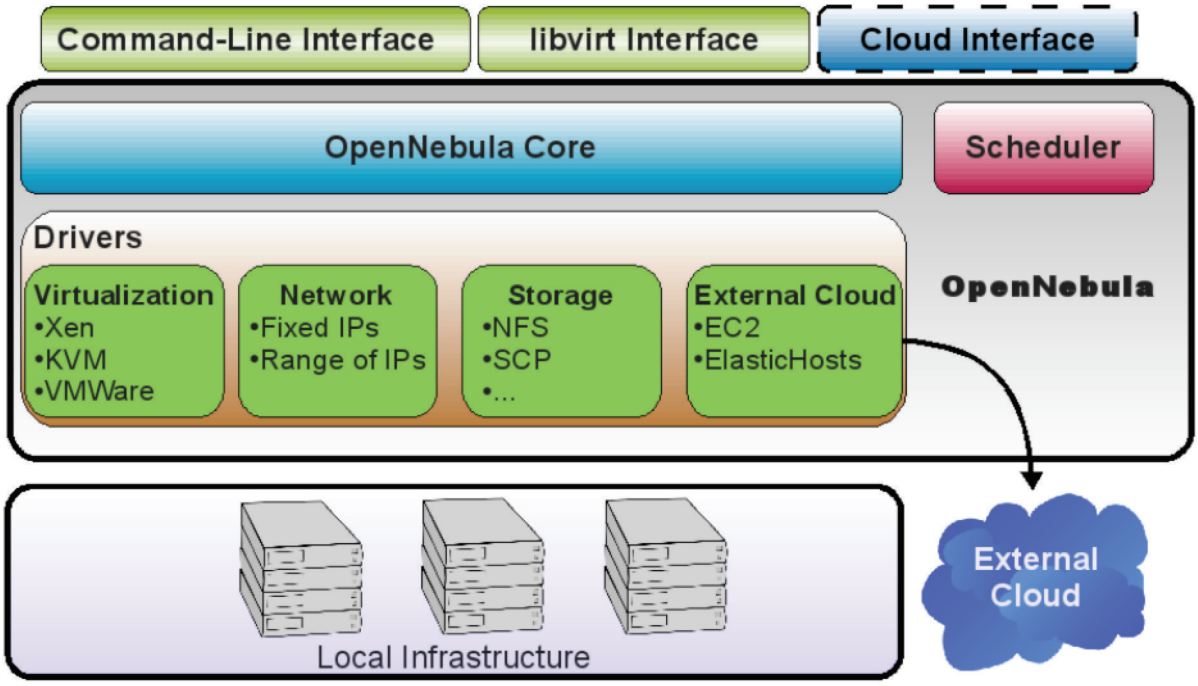

FIGURE 3.3 – Les différents modules d'OpenNebula

3.2.2 Les systèmes autonomes d'ordonnancement de VM

L'application de politiques d'ordonnancement par l'administrateur dans une infrastructure de taille conséquente est difficile voir impossible suivant la stabilité des consommations de ressource. En effet, malgré la visualisation des valeurs de consommations CPU, la mise en place d'une politique d'équilibrage de charge dans une infrastructure de plusieurs centaines de VM peut vite devenir une tâche complexe.

Les ordonnanceurs de VM autonomes (VMS, Virtual Machine Scheduler en anglais) permettent la mise en place et le maintien de politiques d'ordonnancement dans les infrastructures virtualisées. Ces ordonnanceurs reprennent le principe de l'informatique autonome (Autonomic Computing en anglais) [Inc06] selon lequel une application se gère sans intervention humaine, dans notre cas, l'administrateur. Les ordonnanceurs de VM autonomes fonctionnent en quatre étapes : observation, analyse, planification et exécution (MAPE-K). La phase d'observation (M, Monitoring) permet au VMS de récupérer l'organisation des ressources à un instant t. Les données sont ensuite analysées (A, Analyze) en prenant en compte une base de connaissances (K, Knowledge). La base de connaissances des ordonnanceurs est définie par la politique d'ordonnancement éventuellement complétée par des règles de placement. Le résultat de l'analyse est un plan de reconfiguration (P, Plan), c.-à-d. une séquence de reconfigurations, à exécuter (E, Execution).

De nombreux systèmes autonomes mettent en place des politiques d'économie d'énergie par consolidation. Suivant les travaux, le VMS essaie de minimiser le nombre de serveurs utilisés pour héberger les VM de l'infrastructure ou de réduire la consommation électrique globale de l'infrastructure [BSDCP10, NS07, RRT+08]. Les deux méthodes semblent être identiques au premier abord mais peuvent différer pour plusieurs raisons :

– tous les serveurs n'ont pas le même profil énergétique. Suivant les composants électroniques,

un serveur consomme plus ou moins d'électricité ;
- l'utilisation de deux petits serveurs peu gourmands en électricité peut être moins coûteux que l'utilisation d'un seul gros serveur.

Quelque soit l'algorithme utilisé, le but est de consolider les VM sur un ensemble optimal de serveurs afin d'éteindre les serveurs non utilisés. Les ressources mises à disposition par les serveurs restant sont alors exploitées au maximum. Cependant, une consolidation trop importante entraîne un fonctionnement dégradé des services s'exécutant sur les VM à cause d'un manque de ressources. Plusieurs travaux [KTK+09, TDC+07, BKB07, KCD+07, DKL+08] mettent en œuvre des politiques de consolidation en veillant à ne pas dégrader les services à l'intérieur des VM. Des métriques comme le SLA (Service Level Agreement en anglais) doivent alors être récupérées afin de les prendre en compte dans le calcul de la consolidation à effectuer. Le but est de trouver un compromis entre l'économie d'énergie et la performance des applications.

Une autre politique d'ordonnancement couramment utilisée est l'équilibrage de charge[RC10]. Le but de cette politique est de répartir la consommation des ressources CPU équitablement sur les serveurs. Cette répartition des ressources permet d'absorber facilement une hausse des consommations car tous les serveurs sont utilisés pour supporter les nouveaux besoins. Cette politique ne permet pas l'extinction de serveurs car tous les serveurs sont constamment utilisés.

Les travaux précédemment cités s'intéressent à l'organisation des ressources CPU et mémoire dans une infrastructure virtualisée. Cependant, d'autres ressources, comme le réseau ou les disques durs, sont aussi gérées par l'administrateur. Les disques durs sont utilisés en informatique pour stocker de l'information de manière permanente. Par exemple, le système d'exploitation est stocké sur le disque dur afin d'éviter une installation systématique après chaque arrêt du serveurs. Les serveurs et les VM utilisent donc régulièrement le disque dur. Dans le cas des VM, les images disque, jouant le rôle de disque dur, sont localisées sur des serveurs contenant une capacité de stockage importante. Ces serveurs sont ensuite partagés par plusieurs hyperviseurs afin de pouvoir facilement créer et migrer des VM d'un hyperviseur à un autre (voir figure 3.4). Un serveur de stockage doit pouvoir fournir un débit d'écriture et de lecture suffisant pour permettre à toutes les VM d'avoir accès à leur image disque afin de ne pas détériorer leur fonctionnement.

Parda [GAW09] propose un système permettant une distribution équitable des ressources disque de VM hébergées par des hyperviseurs utilisant un même serveur de stockage.

La programmation d'algorithmes d'ordonnancement n'est pas triviale [SCKN07]. De nombreuses méthodes de programmation ont été utilisées pour résoudre le problème de répartition de VM sur un ensemble de serveurs en prenant en compte les ressources fournies et consommées par les différents éléments. La suite de cette section présente plusieurs travaux s'intéressant à cette problématique.

La méthode décrite par Leinberger et al. [LKK99] est utilisée pour la consolidation dans les travaux de Borgetto et al. [BSDCP10]. Cette méthode utilise le MCB (Multi Capacity Bin-packing en anglais) qui consiste à trier les serveurs d'après leurs ressources puis à leur affecter les VM de l'infrastructure en prenant soin de respecter les limites de chaque serveur.

Dans les travaux de Das et al. [DKL+08], un approche multi-agents est présentée pour mettre en place une politique d'économie d'énergie tout en respectant des critères de performance (c.-à-d., une certaine qualité de service). Le système est composé de trois agents interagissant entre eux (voir figure 3.5). Un premier agent est responsable de maintenir la qualité de service en mesurant la charge portée par les applications puis en décidant si une ou plusieurs applications doivent être ajoutées ou supprimées. Un deuxième agent est responsable de l'économie d'énergie. Celui-ci observe les consommations électriques des serveurs. De plus, il est capable d'allumer et d'éteindre des serveurs. Un agent de coordination récupère les données sur la qualité de service et sur la consommation électrique et envoie les commandes d'extinction de serveurs ou d'ajout

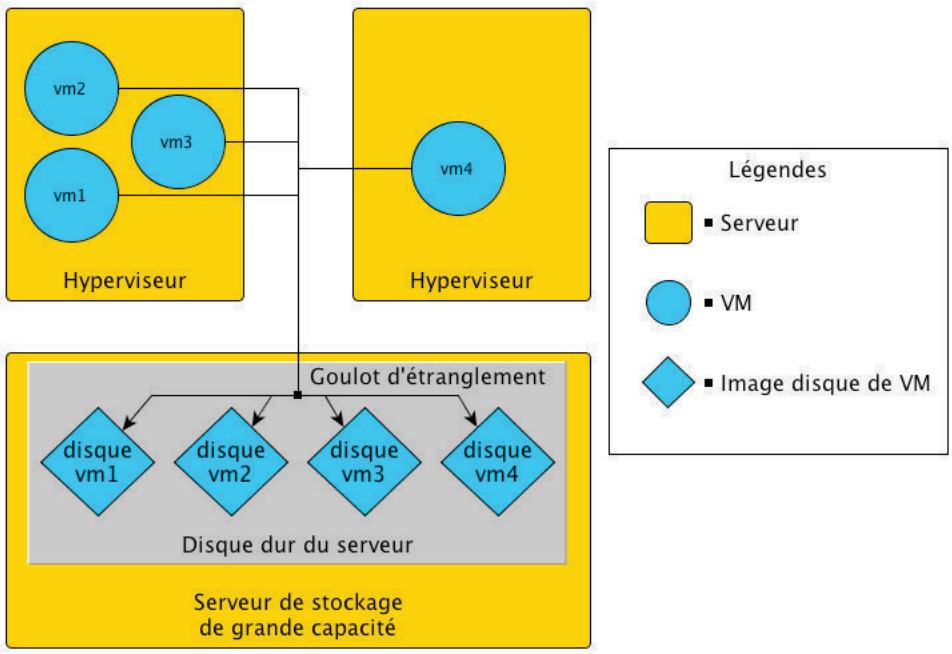

FIGURE 3.4 – Plusieurs hyperviseurs utilisent un même stockage pour faciliter l'administration de leurs VM

d'applications à l'agent correspondant suivant la décision calculée.

Von Der Krogt et al. [VDKFLS10] et Hermenier et al.[HLM10] utilisent la programmation par contraintes (PPC) pour implémenter des algorithmes d'ordonnancement. La PPC [Kot10] est utilisée pour la résolution de problèmes combinatoires de grandes tailles tels que les problèmes de planification et d'ordonnancement. Un des intérêts de la PPC est de séparer la définition du problème, appelé problème de satisfaction, et sa résolution. En effet, le problème de satisfaction est décrit par le programmeur à l'aide d'un langage de haut niveau. Cette description consiste à modéliser l'infrastructure à partir de variables représentant les serveurs et les VM et de définir la politique d'ordonnancement par des relations entre ces variables. Par exemple, les ressources demandées par les VM hébergées sur un serveur doivent être inférieure aux ressources fournies par le serveur en question. La résolution du problème de satisfaction est effectuée par un solveur de contraintes. Le solveur va construire un arbre de solutions et l'explorer à l'aide d'une stratégie de recherche. Cette stratégie permet de trouver une solution, quand elle existe, et de prouver qu'il n'existe pas de solution au problème si le solveur n'a trouvé aucune solution.

Les travaux de Bobroff et al. [BKB07] utilisent la prédiction pour reconfigurer l'infrastructure afin de répondre à des besoins futurs. La prédiction est faite en analysant les ressources consommées sur une période antérieure. Ensuite, une modélisation mathématique de la consommation des ressources est appliquée afin de calculer les consommations futures des VM. Un algorithme d'ordonnancement est ensuite utilisé pour répartir les VM sur les serveurs en prenant soin de répondre aux besoins de chaque VM. Les serveurs non utilisés pendant une période suffisamment longue sont alors éteints. La méthode de prédiction permet d'éviter les effets « ping pong ». En effet, lorsqu'un VMS calcule l'organisation de l'infrastructure en se basant sur les ressources observées, il ne connait pas l'évolution future des ressources. Une VM peut alors être migrée d'un serveur A à un serveur B à un instant t pour être migrée du serveur B au serveur A à l'instant t+1. On parle alors d'effet « ping pong ».

La logique floue [Zad65] a également été utilisée pour la programmation d'algorithmes d'or-

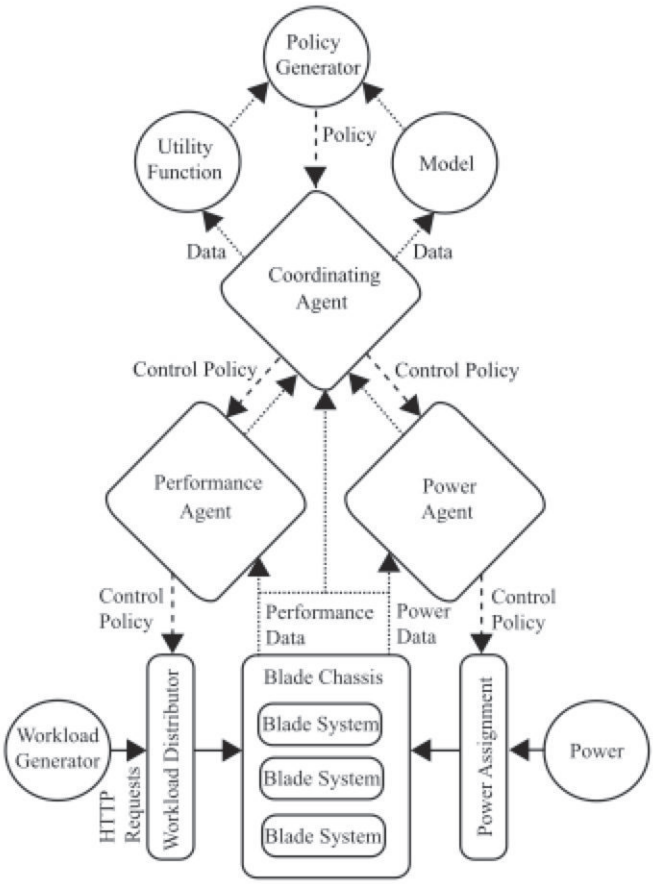

FIGURE 3.5 – Architecture multi-agents pour la mise en place d'une politique d'économie d'énergie

donnancement [XZF+08]. La logique floue est un paradigme de programmation surtout utilisé en intelligence artificielle. Elle s'oppose à la logique booléenne en permettant à une condition d'être dans un autre état que vrai ou faux. Par exemple, considérons qu'il fait chaud si la température est supérieure à 30 °C et qu'il fait froid si la température est inférieure à 10 °C. La logique booléenne ne fait pas la différence entre une température de 12 °C et une température de 28 °C, elles sont toutes les deux ni froides ni chaudes. La logique floue permet de faire la différence entre ces deux températures en précisant que 12 °C est plutôt froid et 28 °C est plutôt chaud. De même, la logique booléenne ne fait pas la différence entre -15 °C et 8 °C alors que la logique floue désigne -15 °C comme une température très froide et 8 °C comme une température froide. L'algorithme d'ordonnancement présenté par Xu et al. utilise la logique floue afin de prédire les ressources consommées et d'adapter l'organisation de l'infrastructure en conséquence. La prédiction est donc faite à partir des données observées, contrairement aux travaux précédemment cités. Aucune analyse des consommations passées est donc nécessaire.

Le procédé de « vol de travail » (work stealing en anglais) décrit par Blumofe et Leiserson [BL94] a été adapté pour mettre en place une politique d'équilibrage de charge [RC10]. Dans ces travaux, l'algorithme d'ordonnancement est mise en place dans une infrastructure pair-à-pair. Chaque serveur surveille ces propres ressources. Quand un serveur détecte un manque de ressources, il met en partage une ou plusieurs de ses VM. Dans le même temps, les serveurs disposant de suffisamment de ressources disponibles vont essayer de « voler » des VM aux serveurs surchargés.

La programmation génétique (PG) [Koz90] a été appliquée par Stillwell et al. [SSVC10] dans le but de résoudre des problèmes d'ordonnancement de VM. La programmation génétique est inspirée du mécanisme de sélection naturelle décrit par Charles Darwin. Le principe de la PG est de partir d'une population initiale composée d'individus pour ensuite la faire évoluer afin d'améliorer les individus la composant et de converger vers la solution d'un problème. Chaque individu représente une solution potentielle au problème à résoudre. Un individu est composé de plusieurs gènes représentant chacun une partie de la solution. L'évolution de la population subit plusieurs générations et chaque génération comprend trois phases : la phase de reproduction, la phase de mutation et la phase de sélection. La phase de reproduction consiste à diviser chaque individu en plusieurs parties pour ensuite assembler ces parties pour former de nouveaux individus. Les individus nouvellement créés sont alors ajoutés à la population. La phase de mutation permet de créer de nouveaux individus. Tout ou partie de la population subit une mutation qui se traduit par une modification aléatoire d'un ou plusieurs de ces gènes. Finalement, une sélection des individus est effectuée afin d'éviter une croissance de la population qui entraînerait des temps de calcul importants. Chaque individu est donc évalué et les meilleurs individus sont conservés pour la prochaine génération. Suivant les algorithmes, quelques individus différents peuvent être ajoutés à la population pour éviter les optimums locaux. Un optimum local est à opposer à un optimum global (voir figure 3.6). Dans une recherche de solution, l'optimum global est la meilleure de toutes les solutions alors que l'optimum local est la meilleure des solutions pour un sous-ensemble des solutions.

Lors de l'application de la programmation génétique aux algorithmes d'ordonnancement 3.7, les différents éléments sont définis comme suit :

- la population initiale est obtenue à partir de l'individu, représentant l'organisation courante de l'infrastructure, additionné à des individus générés aléatoirement ;
- un individu représente une organisation de l'infrastructure ;
- un gène correspond soit à un serveur soit à une VM. Dans le premier cas, la valeur du gène dépend des VM que le serveur héberge. Dans le second cas, la valeur du gène dépend du serveur qui héberge la VM ;

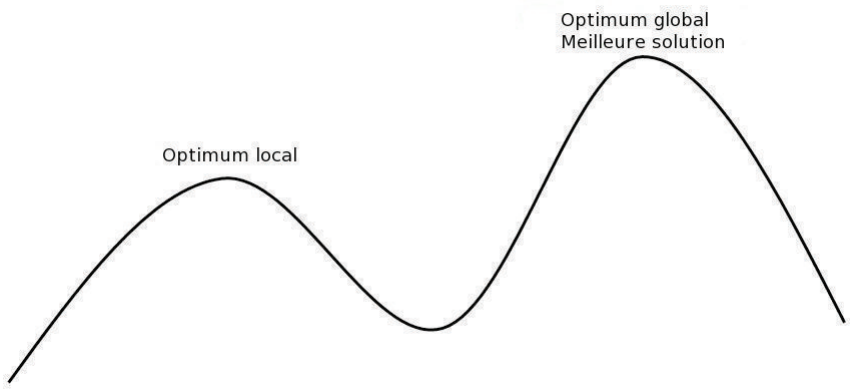

FIGURE 3.6 – Représentation de l'optimum local

- la reproduction de deux individus correspond à migrer plusieurs VM. Une phase de réparation est ensuite indispensable pour corriger les erreurs (les VM hébergées par deux serveurs et les VM n'étant pas hébergées ;
- une mutation correspond à une migration d'une VM d'un serveur à un autre ;
- l'évaluation des organisations dépend de la politique d'ordonnancement voulue. Pour une consolidation, le score d'un individu sera proportionnel au nombre de serveurs non utilisés.

Les algorithmes d'ordonnancement cherchent donc à améliorer l'organisation des ressources de l'infrastructure d'après une politique définie par l'administrateur. Le résultat de ces algorithmes est un plan de reconfiguration composé en grande partie de migrations afin de réorganiser l'infrastructure sans arrêter les services s'exécutant sur les VM. Cependant, une migration requiert des ressources réseau, pour transférer la mémoire d'un serveur à un autre, et cpu, les hyperviseurs effectuent des opérations lors d'un envoi ou d'une réception de VM. Cette problématique [HLM10] doit être prise en compte afin d'éviter des plans de reconfiguration gourmands en ressources et donc dangereux pour la stabilité de l'infrastructure.

Les travaux ci-dessus proposent une gestion des VM en migrant les VM manquant de ressources sur des serveurs possédant plus de ressources disponibles. Cependant, les ressources d'une VM sont définies par le nombre de périphériques virtuels dont elle dispose. Par exemple, le nombre de VCPU modifie la quantité de ressource CPU utilisée par une VM. Les périphériques virtuels doivent être définis lors du démarrage de la VM afin que l'hyperviseur lui affecte les ressources demandées. L'amélioration récente des hyperviseurs permet dorénavant de modifier les périphériques virtuels durant l'exécution de la VM et donc d'augmenter les ressources qui lui sont affectées. La modification des périphériques virtuels d'une VM peut alors être utilisée afin d'augmenter la qualité de service d'une application sans faire de migration [WSVY09].

Les politiques d'ordonnancement décrivent l'organisation globale de l'infrastructure. Néanmoins, l'administrateur doit pouvoir ajouter des règles d'ordonnancement précises sur quelques serveurs ou VM sans modifier la politique mise en place. Par exemple, l'infrastructure est gérée par une politique d'économie d'énergie et trois VM sont créés afin d'effectuer le même service. Pour des raisons de sécurité, l'administrateur doit pouvoir exprimer le fait que ces VM ne doivent pas être hébergées par le même serveur. En effet, si le serveur tombe en panne, le service n'est plus accessible. Pour répondre à des besoins précis, l'administrateur doit pouvoir ajouter des règles de placement à une politique d'ordonnancement. Entropy [HLM10] et Nefeli [TRFD10] proposent tous deux la gestion de règles de placement permettant à l'administrateur de spécifier des relations entre VM et entre VM et serveurs.

FIGURE 3.7 – Une génération d'un algorithme générique

VMware vSphere avec le module DRS (Dynamic Resource Scheduler) [VMW06] propose un ordonnanceur autonome de VM prenant en charge des règles de placement. Ces règles sont accessibles via l'interface graphique et peuvent être activées, désactivées, supprimées ou modifiées.

Les ordonnanceurs de VM autonomes permettent de maintenir une politique d'ordonnancement plus ou moins configurables suivant l'outil utilisé et la présence ou non de règles de placement. Ces outils influencent fortement l'administration de l'infrastructure en garantissant l'organisation correcte des ressources et donc la stabilité de l'infrastructure. Une panne du serveur hébergeant l'ordonnanceur peut donc causer des troubles importants de fonctionnement des VM et de leurs applications. De plus, le temps de calcul d'un plan de reconfiguration pour une infrastructure de taille conséquente à partir d'un seul serveur peut être très long et donc inutilisable. Pour pallier le problème de pannes et de passage à l'échelle, des approches décentralisées [ACJ08, QL11b, CA11] d'ordonnanceurs autonomes ont vu le jour.

Les ordonnanceurs autonomes prennent en charge l'organisation des ressources de l'infrastructure sans intervention de l'administrateur. L'administrateur ne peut donc choisir ni quand appliquer le plan de reconfiguration, ni sur quels serveurs la politique d'administration doit être effective. Les systèmes autonomes sont conçus pour répondre à un besoin précis, par exemple, appliquer une politique d'économie d'énergie. Le détail des reconfigurations appliquées est alors souvent caché à l'administrateur qui ne sait pas pourquoi des reconfigurations ont lieu. De plus, l'administrateur n'a pas accès à la représentation de l'infrastructure qui est construite par l'ordonnanceur, il lui est donc difficile de détecter les comportements anormaux de serveurs ou de VM comme une consommation excessive de CPU. Ces systèmes apportent une solution à un problème précis résolu par la mise en place d'une politique d'ordonnancement définit par l'outil utilisé. Il est donc difficile pour l'administrateur de modifier le comportement du système autonome afin de résoudre un problème observé.

3.3 Les langages dédiés

L'utilisation d'un langage pour administrer une infrastructure de taille conséquente fournit une alternative intéressante aux ordonnanceurs et aux gestionnaires de VM. En effet, l'utilisation d'un langage permet à l'administrateur l'utilisation de scripts et la manipulation d'ensemble d'éléments de grande taille. Le principal inconvénient de l'utilisation d'un langage est l'apprentissage du langage qui est moins intuitif qu'une interface graphique.

Dans cette partie, après une courte introduction sur les langages dédiés, différents langages dédiés à l'administration sont présentés. Ces langages ont été classés d'après trois sous domaines de l'administration : la description de l'infrastructure, la sélection d'éléments au sein d'une infrastructure et la reconfiguration d'une infrastructure virtualisée.

3.3.1 Définition

Les langages dédiés (DSL, Domain Specific Language en anglais) [vDKV00] sont conçus pour être utilisés dans un domaine précis. Ils sont donc à opposer aux langages généralistes qui permettent l'écriture d'applications dans de nombreux domaines (par ex., bancaire, scientifique, linguistique, etc.). Le programmeur a le choix entre plusieurs langages généralistes pour écrire son application en fonction de ses préférences ou des caractéristiques d'exécution du langage. Par exemple, le langage Java permet la création d'applications portables, c'est-à-dire, pouvant s'exécuter sur des systèmes d'exploitation différents, ce qui n'est pas le cas avec le langage C. Contrairement à ces langages, les langages dédiés sont conçus pour résoudre les problèmes d'un domaine précis. Le langage SQL (Structured Query Language, en anglais) sert à effectuer des

opérations sur les bases de données. Le langage HTML (HyperText Markup Language, en anglais) est un langage dédié à l'écriture de pages web. MATLAB (MATrix LABoratory, en anglais) est un langage utilisé dans le domaine scientifique à des fins de calculs numériques. Ces langages ont été conçus pour un domaine précis et ne peuvent pas être utilisés dans un autre domaine.

Le but premier d'un langage dédié est d'augmenter l'efficacité et la fiabilité des tâches effectuées à l'aide du langage. Ces propriétés, intrinsèques au langage, sont apportées par la collaboration d'un programmeur et d'un expert du domaine lors de la conception du DSL. La fiabilité du DSL apporte à l'utilisateur plus de sécurité en l'empêchant de commettre des erreurs. Par exemple, le langage Bossa [BM02] est un langage dédié à l'écriture de politique d'ordonnancement de processus dans les systèmes d'exploitation. Bossa propose à l'utilisateur un langage de haut niveau facilitant l'intégration de nouvelles politiques d'ordonnancement tout en empêchant la perte de processus. L'efficacité du DSL permet à l'utilisateur de décrire la tâche à effectuer de manière succincte et améliore le temps d'exécution de cette même tâche. Un DSL est, par définition, léger, c'est-à-dire composé de peu d'idiomes. En effet, la restriction du langage à un domaine précis limite la taille du langage. Le langage s'apprend donc plus rapidement. De plus, la participation d'un expert du domaine dès la conception du langage permet la spécification d'une syntaxe proche des outils utilisés dans le domaine en question. Le DSL est donc plus intuitif qu'un langage généraliste, surtout si les utilisateurs ne sont pas des informaticiens.

Le principal inconvénient d'un langage dédié est l'apprentissage du langage par les utilisateurs. De plus, le langage étant spécifique à un domaine, un DSL est utilisé par moins de personnes qu'un langage généraliste. Il peut alors s'avérer difficile de trouver de l'aide ou des exemples du langage. La restriction du DSL à un seul domaine peut engendrer une prolifération de langages dédiés. Un utilisateur travaillant dans plusieurs domaines doit alors se familiariser avec plusieurs langages.

3.3.2 Les langages de description d'infrastructures

Certains grands groupes comme Facebook, Google possèdent plusieurs centres de données répartis dans plusieurs villes voire plusieurs pays. Nous rappelons que, par le terme « infrastructure », on ne désigne pas seulement un seul centre de données mais éventuellement une fédération de centres de données.

Plusieurs travaux abordent la description d'infrastructures afin de spécifier des ressources à utiliser ou à configurer.

Puppet [Kan06] est un outil permettant la description de tâches administratives dans le but de les automatiser afin de faciliter et de sécuriser le déploiement d'applications et de serveurs. Les ressources sont décrites par l'ajout de propriétés à des éléments assurant une configuration correcte. Par exemple, le fichier de configuration du démon SSH est décrit de la façon suivante :

```
file { 'sshdconfig':
      path => $operatingsystem ? {
        solaris => '/usr/local/etc/ssh/sshd_config',
        default => '/etc/ssh/sshd_config',
      },
      owner => 'root',
      group => 'root',
      mode  => '0644',
    }
```

La description ci-dessus spécifie l'emplacement du fichier suivant le système d'exploitation et les droits d'accès nécessaires pour le bon fonctionnement de l'application d'accès à distance.

Tune [CBC+08] propose un langage s'inspirant de la méthode UML (Unified Modeling Language, en anglais) pour décrire des architectures composées de plusieurs applications. Les interactions entre les applications sont décrites ainsi que certaines opérations comme le redémarrage ou le déploiement d'une architecture. L'administrateur gère alors des groupes d'applications plus efficacement grâce à des procédures formellement définies.

Tune et Puppet facilitent le déploiement et la maintenance d'une infrastructure. Cependant, la description des ressources n'est pas étudiée dans le but de représenter un ensemble important de serveurs. Les ressources offertes par les serveurs ne sont donc pas présentes dans la description.

VXDL [GPKC09] propose la description d'infrastructures virtualisées en prenant en compte la topologie du réseau physique et les connexions entre les VM. Les serveurs sont décrits d'après leurs capacités disque, réseau (bande passante et latence) et mémoire.

```
group (Cluster_Rendu) {
  function computing
  size (10, 30)
  resource (Node_Cluster_Rendu) {
    function computing
    parameters cpu_frequency (1.6GHz, 3GHz),
    memory_ram (512MB, 2GB)
  }
}
```

VXDL s'intéresse aux interconnexions réseau entre les différents serveurs et VM présents dans l'infrastructure. La description de l'organisation des serveurs au sein de l'infrastructure n'est donc pas présente. Pour des besoins d'administration, l'administrateur doit avoir une vue globale de l'organisation des serveurs. Par exemple, si une panne est détectée sur un serveur, l'administrateur doit pouvoir le localiser dans le centre de données.

JSDL (Job Submission Description Language) [Anj05] est un langage de description de tâches s'exécutant dans des VM. Les descriptions utilisent la syntaxe XML (Extensible Markup Language, en anglais). L'avantage du XML est de garantir une description correcte à partir des schémas XML vérifiant la structure et les attributs utilisés. La description de la tâche prend en compte les ressources matérielles (CPU, mémoire) mais aussi les ressources applicatives requises comme les fichiers.

JSDL permet la description de l'organisation des VM distribuées entre plusieurs tâches. Cependant, l'utilisation du XML rend la description verbeuse et la structure de la description trop rigide. L'administrateur doit s'adapter au modèle fixé par JSDL afin de décrire au mieux son infrastructure qui peut être très différente du modèle proposé. Par exemple, une infrastructure éparpillée dans plusieurs villes ne ressemblera pas à une infrastructure contenant un seul centre de données.

Les langages précédemment cités s'intéressent à la description d'une partie de l'infrastructure afin de la déployer ou de la configurer. Cependant, l'administration d'une infrastructure virtualisée nécessite une description complète et détaillée de l'infrastructure. En effet, un administrateur ne peut gérer les VM si il ne connaît pas leur serveur hôte ou si la description des ressources consommées (par ex., la capacité mémoire des VM) ou des ressources fournies (par ex., la capacité mémoire des serveurs) est incomplète. De plus, la description doit permettre d'ajouter de l'information comme l'organisation des serveurs au sein de l'infrastructure. Par exemple, si l'in-

frastructure comporte deux sites situés dans des villes différentes, l'administrateur doit pouvoir le spécifier afin de pouvoir localiser le serveur.

3.3.3 Les langages de sélection de ressources

Une fois l'infrastructure décrite, l'administrateur va devoir sélectionner des ressources (le plus souvent des serveurs) pour pouvoir les utiliser. Dans une infrastructure de taille conséquente, l'administrateur nécessite des outils afin de sélectionner un ensemble de ressources en décrivant les caractéristiques recherchées. Par exemple, l'administrateur veut 4 serveurs ayant une capacité CPU supérieure à 2 Ghz pour exécuter des calculs scientifiques. Les langages de sélection aident l'administrateur à acquérir rapidement des ressources présentes dans son infrastructure.

Sword [AOVP05] est un langage utilisé pour acquérir des ressources comme des serveurs à partir de requêtes exécutées sur un modèle stocké dans une base de données. Ce modèle est construit à partir d'un système d'observation collectant de nombreuses informations sur les ressources disponibles (voir figure 3.8). Deux types de ressources sont décrites : les ressources nécessaires et les ressources préférées.

```
Group Europe
  NumMachines 4
  Required FreeDisk [300.0, MAX]
  Preferred FreeDisk [1000.0, MAX]
  Required OS [''Linux'']
  Required AllPairs Latency [0.0, 20.0]
  Preffered AllPairs Latency [0.0, 10.0]
```

La requête ci-dessus récupère un groupe de quatre serveurs ayant entre 300 Mo et 1000 Mo d'espace disque disponible et un système d'exploitation « Linux ». De plus, les communications réseau entre les serveurs doivent avoir une latence inférieure à 20 ms.

Le langage VGDL (Virtual Grid Description Language) [CCsK+04] s'intéresse à la description d'une infrastructure dans le but de la déployer à l'aide de l'outil VGES (Virtual Grid Executor System, en anglais) sur des serveurs. Dans l'exemple suivant, un serveur avec une base de données est demandé ainsi qu'une grappe de serveurs avec un minimum d'1 Go de mémoire.

```
EOL2 = SNode ={has databases} far LooseBagOf<CNode>[32:4096];CNode =
{memory >= 1GB}
```

ClassAd [TWML02] est un langage de description prenant en compte de nombreux types de ressources (CPU, réseau, mémoire...) et permettant de classer les serveurs d'après leurs ressources. Dans l'exemple suivant, les serveurs sont sélectionnés à partir de leurs ressources CPU, mémoire et disque et une fonction de classement est définie sur les propriétés du serveur :

```
Disk = 323496; // kbytes
Memory = 64; // megabytes
Arch = "INTEL";
Rank
  member(other.Owner, ResearchGroup) * 10
  + member(other.Owner, Friends);
Constraint =
  !member(other.Owner, Untrusted)
  && Rank >= 10
```

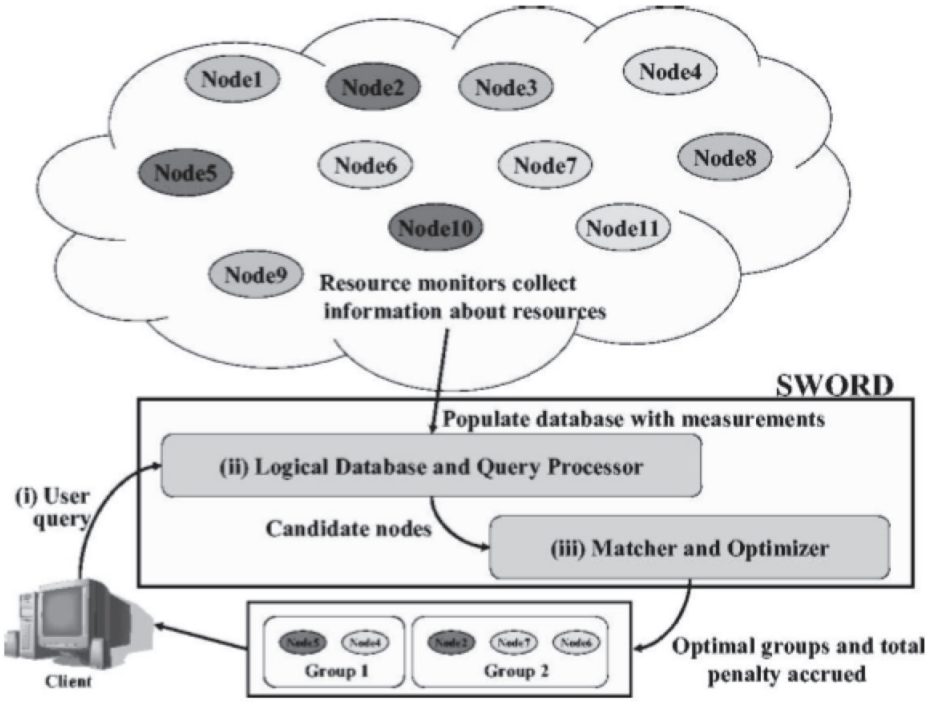

FIGURE 3.8 – Fonctionnement général de l'outil Sword

On remarque que la fonction de classement est aussi utilisée pour limiter le nombre de serveurs sélectionnés.

XRSL (Extended Resource Specification Language, en anglais) [All09] est un langage permettant de décrire les ressources utilisées par une tâche. La description prend en compte des besoins très précis comme les urls ou les fichiers.

Les langages de sélection de ressources permettent de récupérer des ressources dans l'infrastructure d'après des propriétés. Les plus avancés [AOVP05, TWML02] proposent des fonctionnalités comme le classement des serveurs d'après leurs ressources soit par le biais d'une fonction de classement soit par spécification de propriétés préférentielles. Cependant, ces langages ne permettent pas la sélection de ressources d'après l'organisation des serveurs et des VM, par exemple en récupérant tous les serveurs situés à Nantes ou toutes les VM d'un même serveur.

FPath et FScript [DLLC09] sont deux langages complémentaires permettant respectivement la sélection et la reconfiguration d'éléments dans les architectures à composants Fractal. Un composant Fractal est définit par :

- un code métier fournissant une ou plusieurs fonctionnalités ;
- un ensemble de contrôleurs permettant d'agir sur le composant. Par exemple, l'arrêt de fonctionnalités ;
- des interfaces permettant de communiquer avec les autres composants.

De plus, un composant peut être composé d'un ou plusieurs composants. Il est alors appelé « composant composite » et une hiérarchie père/fils est créé. À l'aide du langage FPath, l'architecture est parcourue en sélectionnant les composants et en parcourant leurs interfaces ou la hiérarchie père/fils (voir figure 3.9).

Afin d'affiner la sélection des composants, des filtres prenant en compte les propriétés des composants et leur organisation dans l'architecture Fractal sont utilisés.

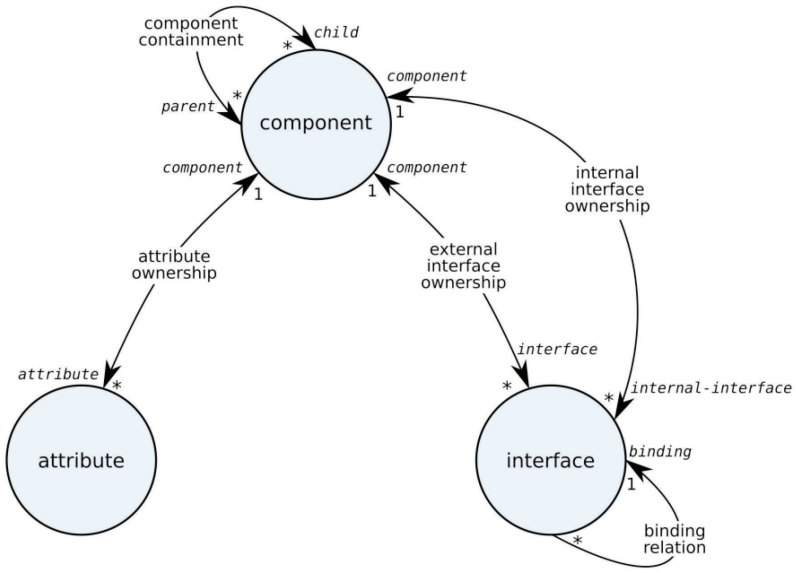

FIGURE 3.9 – Les axes de navigation utilisés par FPath

```
$dest/attribute::*[name(.) == name("mon_composant")]
$root/descendant-or-self::*[size(./parent::*) > 1]
```

Dans la première requête ci-dessus, les composants du composite « $comanche » ayant comme nom « mon_composant » sont sélectionnés. Dans la seconde requête, les composants ayant plusieurs parents sont récupérés. On obtient alors la liste de tous les composants partagés par plusieurs composants.

Les langages dédiés tels que Sword, XRSL, VGDL ou ClassAd s'intéressent à la sélection d'éléments dans une infrastructure à partir de leurs propriétés. Cependant, ces langages ne prennent pas en compte la navigation à partir d'un élément vers d'autres. Cette navigation est essentielle dans le contexte d'infrastructure virtualisée afin d'obtenir tous les serveurs d'un site ou toutes les VM d'un serveur. FPath offre un langage permettant la navigation et la sélection précise d'éléments dans une architecture à composants Fractal. Cependant, les concepts d'interfaces et de contrôleurs rendent le langage verbeux. Ces concepts n'étant pas présents dans une infrastructure virtualisée, il serait nécessaire de disposer d'une adaptation plus concise de FPath.

3.3.4 Les langages de reconfiguration d'infrastructures virtualisées

Les premiers langages utilisés pour la reconfiguration d'infrastructures virtualisées ont été les interpréteurs de commandes (shell) proposés par les hyperviseurs (par ex., xm pour l'hyperviseur Xen) afin de manipuler les VM qu'ils hébergent. Cependant, chaque shell a son propre langage dédié (DSL, Domain Specific Language) offrant des fonctions permettant d'exécuter des reconfigurations sur les VM. Celles-ci permettent la gestion du cycle de vie des VM comme le démarrage, l'arrêt et la migration. L'exemple ci-dessous montre la commande permettant la migration d'une VM « vm1 » sur un serveur « serveur1 », respectivement, à partir d'un hyperviseur Xen et d'un hyperviseur KVM. Le nom du serveur est affecté lors de sa configuration réseau alors que le nom de la VM est affectée par l'hyperviseur lors de la création de la VM. Avec Xen, la gestion de la VM « vm1 » passe par une connexion à l'hyperviseur l'hébergeant. Ensuite, on spécifie la VM à

migrer et le serveur de destination. Le paramètre « -l » permet la migration à chaud. Avec KVM, la migration requiert une connexion à la VM. Une fois la connexion établie, la VM est migrée vers le serveur « serveur1 ».

```
xm migrate vm1 serveur1 -l
migrate tcp:(serveur1):4444
```

La gestion de VM peut donc être très différente d'un hyperviseur à un autre ce qui complique l'administration d'une infrastructure composée d'hyperviseurs différents. La librairie libvirt [Libc] et son shell associé virsh ainsi que l'outil usher [MGVV07] et son shell ush propose une interface de gestion de VM indépendant de l'hyperviseur sous-jacent pour les hyperviseurs les plus courants comme Xen, VMware ESX et KVM. Par exemple, la migration de la VM « vm1 » avec virsh s'écrit :

```
virsh -c serveur0 migrate --live vm1 qemu+ssh://serveur1/system
```

Le paramètre « -c » permet une connexion à l'hyperviseur hébergeant la VM « vm1 ». Ce paramètre peut être omis si la commande est exécutée directement à partir de l'hyperviseur concerné. Ensuite, la migration est effectuée à partir des noms des acteurs et du protocole de connexion utilisé. Ce protocole varie légèrement suivant l'hyperviseur ciblé (voir figure 3.3.4).

protocole	hyperviseur ciblé
qemu+ssh ://serveur1/system	KVM
xen+ssh ://serveur1/system	Xen
esx ://example-esx.com	VMware ESX

TABLE 3.1 – Protocoles utilisés par virsh pour la connexion aux différents hyperviseurs

La librairie libvirt et l'outil Usher proposent la gestion d'un hyperviseur à distance afin d'éviter à l'administrateur de nombreuses connexions. L'administration de l'infrastructure se fait alors à partir d'un seul serveur qui se connecte aux hyperviseurs par l'intermédiaire du VMM.

Les différents shells présentés précédemment propose à l'administrateur des langages dédiés à la manipulation de VM. Cependant, ces langages ne proposent pas de fonctionnalités d'observation permettant de connaître les ressources consommées par une VM. L'administrateur ne peut donc pas savoir quelle VM a besoin de ressources supplémentaires. Ces informations peuvent être récupérées en utilisant les API (Application Programming Language) respectives de libvirt et de Usher via un langage de programmation généraliste comme Java ou C mais elles ne sont pas directement accessibles via les shells proposés. De plus, ces outils ne sont pas conçus dans le but de gérer un nombre important de VM. La sélection de plusieurs VM n'étant pas disponible, une commande correspond à une reconfiguration. Pour arrêter cent VM, il faut donc saisir cent commandes. Les reconfigurations sont alors exécutées séquentiellement. La parallélisation peut être demandée par l'administrateur en exécutant la commande en arrière-plan sur un système UNIX. Dans ce cas, l'administrateur doit veiller au bon déroulement des reconfigurations.

Plasma [HLMM11] est un langage de reconfiguration déclaratif injectant des règles de placement dans l'ordonnanceur de VM autonome, Entropy. L'administrateur décrit l'organisation qu'il souhaite obtenir et Entropy calcule les reconfigurations nécessaires à l'obtention du placement décrit. À partir de la règle « on » forçant une VM à s'exécuter sur un serveur, des

migrations peuvent donc être effectuées. Ce langage permet la description rapide d'ensembles de VM et de serveurs d'après leur nom. Par exemple, le script suivant définit trois groupes de VM. L'utilisation de l'opérateur « spread » garantit que les VM de chaque groupe ne peuvent pas être hébergées par un même serveur.

```
$T1 = { VM1, VM2, VM3 };
$T2 = VM[4..7];
$T3 = VM[8..9];

spread ($T1);
spread ($T2);
spread ($T3);
```

Plasma est un langage dédié à l'écriture de règles de placement. Par conséquent, aucun mécanisme d'introspection ou de reconfiguration n'est disponible. L'administrateur ne peut donc ni connaître la consommation des ressources sur l'infrastructure ni modifier l'organisation des ressources, par exemple, en migrant une VM.

3.4 Conclusion

Avec l'émergence d'infrastructures virtualisées de taille conséquente, des outils d'administration sont nécessaires afin d'assister l'administrateur. La taille des infrastructures étant en constante hausse, les gestionnaires de VM ne permettent plus une administration efficace nécessitant l'utilisation de scripts et facilitant la manipulation d'ensembles d'éléments. Les ordonnanceurs de VM autonomes mettent en place une politique d'ordonnancement précise sans intervention de l'administrateur mais leur rigidité empêche l'administrateur de les adapter à ses besoins. Des langages dédiés à l'administration d'infrastructures proposent une alternative permettant de bénéficier de l'utilisation des scripts et de la manipulation d'ensemble d'éléments. Cependant, ces langages se focalisent sur une partie de l'administration (déploiement de l'infrastructure, sélection d'éléments, reconfiguration de l'infrastructure). L'administrateur doit alors apprendre plusieurs langages n'ayant pas ou peu de points communs. De plus, les langages dédiés à la reconfiguration d'infrastructures virtualisées présentés ci-dessus ne permettent pas l'introspection, c.-à-d., la récupération des ressources consommées. Ces langages, n'incluant pas le suivi de l'évolution des ressources, ne peuvent offrir à l'administrateur les outils garantissant une organisation cohérente des ressources au sein de l'infrastructure.

Une interaction entre langages dédiés à l'administration et systèmes autonomes doit donc être mise en place afin d'assurer, respectivement, une manipulation efficace d'un nombre conséquent d'éléments et un maintien de la cohérence dans une infrastructure virtualisée de taille conséquente.

À partir des différents travaux étudiés, ce document propose un outils dédié à l'administration d'infrastructures virtualisées de taille conséquente (allant jusqu'à plusieurs milliers de VM). Les interactions entre l'administrateur et l'outil d'administration sont gérés à l'aide de langages dédiés. Ces langages permettent à l'administrateur de décrire, analyser, gérer et optimiser son infrastructure. L'environnement d'exécution de ce langage et l'architecture de l'outil d'administration sont conçus afin de rendre possible la gestion de milliers de VM dans une infrastructure hétérogène (c.-à-d., composée d'hyperviseurs différents). Ces deux aspects ainsi que leur évaluation sont présentés dans la prochaine partie.

Deuxième partie

Contribution

Chapitre 4

Langages dédiés à l'administration d'infrastructures virtualisées

Le nombre conséquent d'éléments à gérer dans une infrastructure virtualisée limite l'utilisation d'interfaces graphiques très répandues dans les gestionnaires de VM. Un approche langage permet la gestion d'ensembles d'éléments importants et l'utilisation de scripts utilisés pour décrire une séquence de reconfigurations. À partir de deux langages dédiés, l'outil d'administration propose la gestion des différentes étapes de l'administration. Le premier langage est dédié à la description concise d'une infrastructure virtualisée. Cette description détaille l'organisation des ressources de l'infrastructure comme l'électricité, les ressources fournies par les serveurs et les ressources consommées par les VM (CPU, mémoire...). Le second langage est dédié à la gestion de l'infrastructure. Ce langage permet la sélection d'ensemble d'éléments facilitant la gestion d'un grand nombre de serveurs et de VM, l'introspection des ressources de l'infrastructure ainsi que la reconfiguration de l'infrastructure agissant sur l'organisation des ressources. Le langage est aussi composé de règles de placement afin de maintenir la cohérence de l'infrastructure en définissant des relations entre VM et entre VM et serveurs. Pour finir, l'outil d'administration propose un opérateur de maintien de la cohérence capable d'utiliser des algorithmes de placement pour appliquer une politique d'administration en résolvant les problèmes de surcharges et les règles de placement non respectées.

Sommaire

Les infrastructures virtualisées tendent à devenir de plus en plus grandes en termes de nombre de serveurs et de VM. Par exemple, Amazon utilise près d'un demi million de serveurs pour son offre d'informatique dans les nuages. La gestion d'un grand nombre de serveurs et, a fortiori, de VM est difficilement concevable à partir de gestionnaires de VM utilisant une interface graphique. Par exemple, la recherche d'une VM dans un centre de données comprenant des centaines de serveurs peut s'avérer compliquée. De plus, l'administrateur voulant éteindre une centaine de VM ne doit pas être obligé de répéter cent fois les mêmes opérations. Pour ces raisons, l'outil d'administration proposé est basé sur des langages dédiés. En effet, l'approche langage permet la manipulation d'ensembles d'éléments et l'utilisation de scripts utilisés pour sauvegarder une séquence de reconfigurations afin de l'exécuter à plusieurs reprises. Ces langages manipulent les éléments de l'infrastructure à partir du nom que leur donne l'administrateur. L'utilisation d'un nom pour identifier de façon unique les éléments d'une infrastructure est courante dans le domaine de l'administration. Par exemple, virsh (le shell de libvirt), ush (le shell de Usher) et VMware vSphere fonctionnent sur ce principe.

Les langages présentés dans ce chapitre s'intéressent plus particulièrement aux infrastructures :

- de taille conséquente car les infrastructures de plusieurs centaines de serveurs hébergeant plus d'un millier de VM sont de plus en plus courantes ;
- multi-hyperviseur car les infrastructures de grande taille sont régulièrement composées de plusieurs hyperviseurs. Il n'est pas rare d'utiliser des hyperviseurs de la communauté « Open Source » comme KVM et des hyperviseurs commerciaux comme VMware sur la même infrastructure ;
- distribuées car la taille grandissante des infrastructures imposent des centres de données avec des localisations différentes. Par exemple, Facebook possède deux centres de données aux États-Unis (un dans l'état de Virginie et un autre en Californie) et un centre de données en Suède.

Deux langages dédiés ont été conçus afin de prendre en charge la totalité de l'administration : un langage dédié à la description de l'infrastructure et un langage dédié à la gestion de l'infrastructure. La description de l'infrastructure permet à l'administrateur d'ajouter des informations sur l'organisation des ressources à administrer. Un intérêt particulier est porté sur la flexibilité de cette description qui ne doit pas restreindre l'administrateur dans l'organisation des différents éléments à administrer comme les serveurs. Cette description est utilisée à l'initialisation de l'outil d'administration afin de construire une représentation de l'infrastructure qui sera manipulée à l'aide du second langage, dédié à la gestion de l'infrastructure. La gestion de l'infrastructure est elle-même divisée en deux parties. La première partie s'occupe de l'introspection et de la reconfiguration de l'infrastructure. La flexibilité apportée par la couche de virtualisation permet à l'administrateur de reconfigurer l'infrastructure sans éteindre ou suspendre l'exécution des VM. Cependant, afin de maintenir une infrastructure cohérente, des règles doivent être mises en place par l'administrateur afin d'éviter les reconfigurations hasardeuses. La deuxième partie de la gestion de l'infrastructure s'intéresse au maintien de la cohérence dans l'infrastructure à l'aide de règles de placement ajoutées par l'administrateur.

4.1 Un langage dédié à la description d'infrastructures virtualisées

La description de l'infrastructure est utilisée pour construire une représentation de l'infrastructure interne à l'outil d'administration. Cette représentation est appelée « le modèle ». Ce modèle comprend de nombreuses informations sur l'organisation des ressources dans l'infrastructure. Ces informations seront ensuite utilisées par le langage de gestion de l'infrastructure afin de sélectionner efficacement des éléments et de faciliter la reconfiguration de l'infrastructure. Ce modèle doit donc être le plus complet possible afin de simplifier les tâches d'administration.

Deux types d'information sont décrits par l'administrateur : l'organisation des ressources et les propriétés statiques de ces ressources. L'organisation des ressources décrit l'architecture globale de l'infrastructure. Les éléments principaux d'une infrastructure virtualisée sont les serveurs et les VM. Cependant, une architecture d'un millier de serveurs hébergeant plusieurs milliers de VM s'organise à l'aide de containers plus ou moins importants suivant la taille et le rôle de l'infrastructure. Par exemple, l'infrastructure d'une grande entreprise est répartie dans plusieurs centres de données, éventuellement localisés dans des villes différentes. La prise en compte de cette information de localisation dans le modèle aide l'administrateur à manipuler les serveurs en sélectionnant, par exemple, tous les serveurs d'un centre de données. Le même procédé est valable pour l'organisation des VM. Pour une plateforme d'« IaaS », les VM sont louées à des clients. L'administrateur peut alors avoir à manipuler toutes les VM d'un client dans le but d'effectuer une tâche d'administration précise. L'affectation d'un utilisateur à chaque VM lors de sa création autorise alors l'administrateur à récupérer rapidement toutes les VM de l'utilisateur. De plus, l'affectation étant gérée par l'outil d'administration, l'administrateur est assuré de sélectionner toutes les VM de l'utilisateur et donc d'effectuer la reconfiguration sans oublier une ou plusieurs VM.

Le deuxième type d'informations décrites dans le modèle sont les propriétés statiques des ressources. Ces propriétés servent à ajouter des précisions sur une ressource. Par exemple, il est courant au sein d'une même infrastructure d'utiliser des serveurs possédant un hyperviseur et d'autres serveurs fournissant des services comme, par exemple, un serveur DHCP gérant les adresses IP des serveurs et des VM de l'infrastructure. Une propriété « rôle » peut alors être ajoutée afin de différencier les hyperviseurs des autres serveurs. Ce type d'informations est aussi utilisé afin de préciser les ressources statiques des serveurs et des VM. Pour un serveur, les ressources statiques sont déterminées par ses composants matériels. Ces ressources sont considérées statiques bien que l'administrateur puisse les modifier en changeant les composants intrinsèques au serveur, par exemple, en ajoutant de la mémoire. Cependant, ces modifications sont rares et nécessitent l'arrêt du serveur. Un outil d'observation peut alors être utilisé afin de vérifier les ressources statiques à chaque démarrage d'un serveur. Pour une VM, les ressources statiques correspondent aux ressources qui lui sont attribuées lors de sa phase d'instanciation. Par exemple, la taille de l'image disque pourra être précisée.

Le modèle décrit l'organisation de l'infrastructure et les propriétés statiques des éléments la composant. Afin de construire ce modèle, l'administrateur décrit l'infrastructure à l'aide de fichiers texte. Décrire une infrastructure de plusieurs centaines de serveurs et VM peut être une opération longue et laborieuse. Le langage dédié à cette tâche s'efforce à décrire synthétiquement une infrastructure. La description de l'infrastructure est composée de trois parties : la séparation de l'infrastructure en plusieurs vues, la description des éléments de chaque vue puis la description des ponts permettant de relier les vues entre elles.

4.1.1 Les vues

Une infrastructure comme un centre de données doit être perçue sous différents angles [KK09] afin d'en faciliter son administration. En effet, l'administration des serveurs d'une infrastructure nécessite la coordination de plusieurs corps de métier suivant les ressources à organiser. Une vue représente alors le regard d'un administrateur sur une ressource particulière. Par exemple, un serveur d'un centre de donnée peut être considéré comme un consommateur d'électricité et comme un producteur de ressources (CPU et mémoire).

Dans le premier cas, l'administration se focalise sur la gestion des alimentations des serveurs. L'alimentation d'un serveur fournit l'électricité requise par tous les composants de ce dernier. Sur les ordinateurs domestiques, en cas de panne de l'alimentation, l'ordinateur s'éteint brutalement. Cet arrêt brutal met, évidemment, fin à toutes les applications en cours d'exécution avec des possibilités de pertes de données et peut aussi endommager des composants. Afin d'éviter ce problème, les serveurs présents dans les centres de données disposent de deux alimentations - la probabilité d'une panne simultanée des deux composants étant pratiquement nulle. Afin d'éviter les pannes électriques, de nombreux centres de données disposent de plusieurs sources d'électricité. Afin d'éviter que l'arrêt d'une source d'électricité arrête les serveurs, chaque alimentation d'un serveur utilise une source différente. En cas de problème sur une des sources, plusieurs serveurs perdent une de leurs alimentations mais continuent à fonctionner à partir de la seconde. Chaque source d'électricité peut alimenter un nombre limité de serveurs. Il faut donc être capable de connaître tous les serveurs reliés à une source afin d'assurer une administration correcte de l'infrastructure.

Lorsque l'administrateur considère les serveurs comme des producteurs de ressources, son regard sur l'infrastructure est différent. En effet, les tâches d'administration vont consister à intervenir sur l'infrastructure afin d'ajouter et de réparer des serveurs. L'administrateur devra donc être capable de localiser précisément l'emplacement d'un serveur. L'organisation du réseau électrique ne l'intéresse plus mais il doit, à présent, connaître le centre de données hébergeant le serveur et l'emplacement exact du serveur dans le centre.

Pour une infrastructure virtualisée, les vues permettent de séparer l'organisation des VM de celle des serveurs. Une vue est alors dédiée à organiser les VM en les regroupant, par exemple, par utilisateur.

Dans les exemples donnés ci-dessus, l'organisation hiérarchique des éléments est utilisée naturellement. Par exemple, un source d'électricité alimente plusieurs disjoncteurs qui sont eux-mêmes reliés à plusieurs prises qui alimentent les serveurs. Les serveurs sont entreposés dans des baies (rack, en anglais) qui sont situées dans des centres de données construits dans des villes. Une vue organise donc des ressources de manière hiérarchique.

4.1.2 Les types

L'organisation hiérarchique d'une vue entraîne une hiérarchisation de ses éléments. Le composite est nommé le père et ses composants sont ses fils. Un centre de donnée est donc le père de multiples serveurs. Afin d'éviter des descriptions d'infrastructures erronées, chaque élément de la vue possède un type. Le typage des éléments permet de limiter les erreurs lors des associations père/fils. Par exemple, un élément de type « utilisateur » peut uniquement être composé d'éléments de type « VM ». Ainsi, si l'administrateur décrit une association entre un « utilisateur » et un « serveur », l'erreur est détectée. De plus, le type ajoute une propriété statique à un élément permettant d'identifier sa place et son rôle dans la vue (voir figure 4.1).

Une vue est alors considérée comme un ensemble de conteneurs. L'élément « racine », seul élément n'ayant pas de père, représente le plus grand des conteneurs stockant l'infrastructure

complète. Chaque fils de cet élément appartient à une vue. Prenons un exemple avec l'organisation physique. L'infrastructure comprend quatre centres de données situés dans deux villes. Chaque centre de données possède des serveurs entreposés dans des baies. Les types de la vue seront alors : « ville », « centre », « baie », « serveur ». Un élément de type « ville » aura comme fils des éléments de types « centre » et un élément de type « serveur » aura comme père un élément de type « baie ».

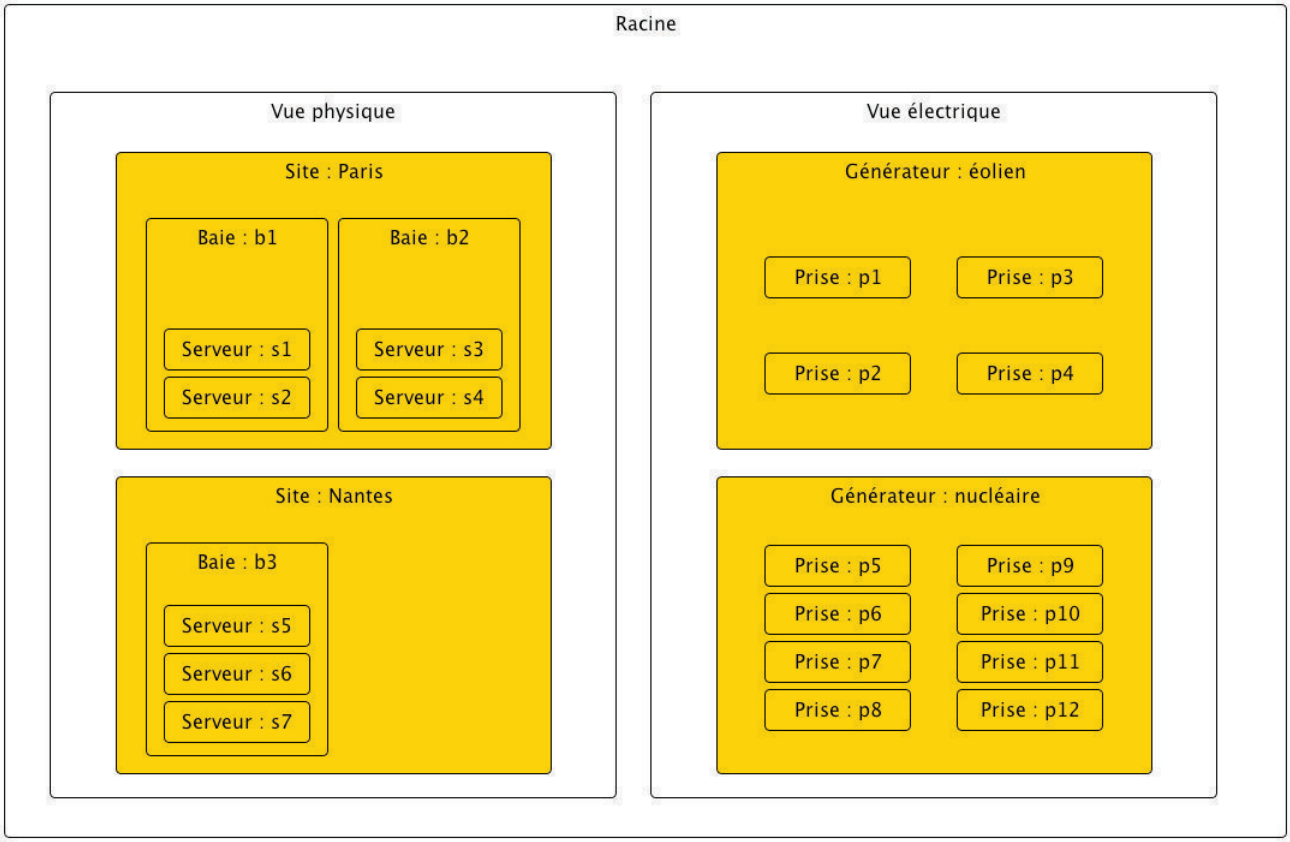

FIGURE 4.1 – Utilisation des types comme conteneurs d'éléments

4.1.3 Les ponts

Chaque vue décrit une organisation de ressources d'après une préoccupation précise. Cependant, ces vues ne sont pas indépendantes. Elles sont liées entre elles afin de permettre l'administration de la totalité de l'infrastructure. Par exemple, lorsqu'un serveur doit être ajouté dans l'infrastructure. L'administrateur doit trouver un ou plusieurs sources d'électricité permettant de l'alimenter. Une fois les sources identifiées, un emplacement dans une baie du centre de donnée doit être trouvé. Le serveur peut alors être démarré et doit être ajouté à la liste des serveurs hébergeant des VM afin d'être utilisé. Les vues doivent donc communiquer afin de pouvoir mettre en relation les éléments des différentes vues. L'administrateur pourra ainsi connaître, par exemple, toutes les VM impactées par une panne d'une des sources d'électricité.

Les ponts permettent l'ajout de relation père/fils entre les vues. Ces ponts sont essentiels afin de relier les différentes vues et de faciliter une administration complète d'une infrastructure virtualisée. Par exemple, la relation d'hébergement entre serveurs et VM sera un pont entre le

type « serveur » et le type « VM ». Une VM sera alors hébergée par un serveur et un serveur aura de multiples VM. Un seul pont est possible entre deux vues.

4.1.4 Un exemple de modélisation d'une infrastructure virtualisée

4.1.5 La syntaxe du langage

La construction du modèle représentant l'infrastructure à administrer est effectuée par lecture de plusieurs fichiers texte. Chaque vue est décrite dans un fichier texte afin de garantir une séparation des préoccupations. Cette séparation est importante dans le but de limiter la taille et la complexité de chaque fichier de description. Les fichiers sont alors plus faciles à écrire, à lire et à modifier.

Dans une vue, la description d'un élément comprend :
- un nom unique permettant son identification ;
- un type ;
- des propriétés ;
- des fils.

Le nom sera utilisé pour identifier et manipuler l'élément dans les tâches d'administration. Le type définit le rôle de l'élément dans l'infrastructure et son emplacement dans la hiérarchie de chaque vue. Les propriétés sont utilisées pour ajouter des informations sur les éléments comme le système d'exploitation installé sur un serveur. Les fils correspondent à tous les éléments contenus par l'élément en cours de description. Par exemple, un centre de données contenant plusieurs serveurs.

La syntaxe pour décrire un élément est (les espaces sont facultatifs) :

```
mon_type : mon_nom
propriete_1 = valeur_propriete
propriete_2 = valeur_propriete
propriete_n = valeur_propriete
fils_1
fils_2
fils_3
```

Les propriétés sont indifféremment définies avant et/ou après les fils. Afin d'appliquer les mêmes propriétés à plusieurs éléments, des modèles peuvent être définis et utilisés comme suit :

```
model : nom_du_modele
propriete_1 = valeur_propriete
propriete_2 = valeur_propriete
propriete_n = valeur_propriete

type_element : element_1
model = nom_du_modele
autre_propriete = valeur

type_element : element_2
model = nom_du_modele
redefinition_propriete = valeur
```

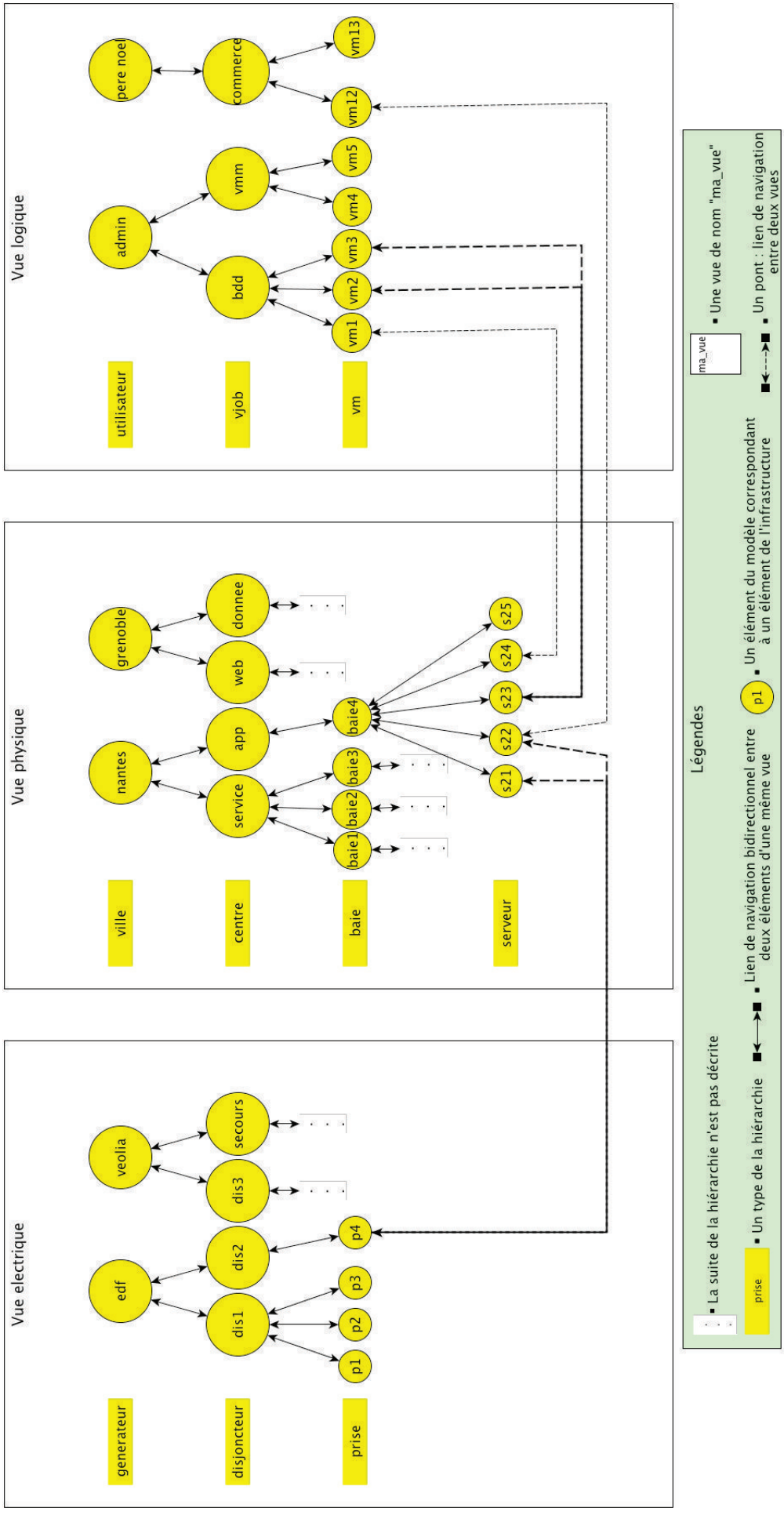

FIGURE 4.2 – Le modèle VM-Script d'une infrastructure contenant trois vues

Les ponts vont créer des liaisons entre les vues. Une liaison relie un élément dit « père » à plusieurs éléments dits « fils ». Par exemple, un serveur sera relié à plusieurs VM afin de décrire une relation d'hébergement. La description d'un pont se fera en réutilisant les noms des éléments définis dans chaque vue :

```
type_du_pere : nom_du_pere
nom_du_fils_1
nom_du_fils_2
nom_du_fils_n
```

Le nom des éléments d'une infrastructure, notamment les serveurs et les VM, comporte souvent une base commune suivi d'un numéro permettant de créer des noms uniques et facilement identifiable. Par exemple, sur la plateforme Grid5000 [Liba], les serveurs sont nommés avec le nom de leur grappe suivi d'un numéro. La grappe nommée 'paradent' possède 64 serveurs nommés « paradent-1 » jusqu'à « paradent-64 ». Dans le but de faciliter la description d'une telle infrastructure, des raccourcis d'écriture sont mises à disposition de l'administrateur. L'opérateur « [] » est disponible pour l'écriture des noms des éléments :

- « serveur[1-3] » sélectionne les éléments nommés « serveur1 », « serveur2 » et « serveur3 » ;
- « serveur[.1,.deux].3 » sélectionne les éléments nommés « serveur.1.3 » et « serveur.deux.3 ».

Le langage dédié à la description d'infrastructures (voir l'annexe 8.2 pour un exemple de description) permet à l'administrateur de spécifier l'organisation des différents éléments à administrer de façon complète. Le langage utilise une syntaxe concise afin de limiter la taille des descriptions. La répartition de la description dans plusieurs fichiers simplifie l'écriture et la modification de cette dernière. La possibilité d'enrichir la description avec des propriétés rend la description plus précise et facilite les différentes tâches d'administration.

4.2 Un langage dédié à l'introspection et à la reconfiguration d'infrastructures virtualisées

Une fois l'architecture décrite, le modèle offre une représentation précise de l'infrastructure virtualisée comprenant l'organisation des ressources, les capacités de chaque ressource, qu'elle soit consommée ou produite, et les informations ajoutées par l'administrateur. Un tel modèle pour une infrastructure de taille conséquente représente une quantité importante d'information. Le langage présenté dans cette partie aide l'administrateur à manipuler les éléments du modèle dans le but de réorganiser les ressources de l'infrastructure.

Trois étapes se distinguent dans la manipulation des éléments de l'infrastructure : la sélection des éléments, l'introspection de l'infrastructure et la reconfiguration de l'infrastructure. La sélection des éléments permet à l'administrateur de choisir les éléments sur lesquels une introspection ou une reconfiguration vont être effectuées. Un intérêt particulier est porté à la sélection d'ensembles d'éléments afin de proposer à l'administrateur une gestion efficace d'une infrastructure de taille conséquente. L'introspection de l'infrastructure permet à l'administrateur d'interroger le modèle dans le but d'obtenir des informations sur les éléments ou de créer des filtres agissant sur la sélection préalablement établie. L'étape de reconfiguration permet à l'administrateur de créer, de supprimer et de modifier les ressources de l'infrastructure. Par exemple, la gestion du cycle de vie d'une VM est effectuée par des reconfigurations.

L'approche de l'administration d'une infrastructure à partir d'un langage dédié offre des fonctionnalités d'administration essentielles, comme l'écriture de scripts et la manipulation de nombreux éléments, peu ou pas supportées par les interfaces graphiques.

4.2.1 Sélection des éléments

4.2.1.1 Unicité du nom des éléments

La sélection d'éléments à partir du langage dédié repose sur l'attribution d'un identifiant unique à chaque élément. Cet identifiant est appelé le « nom » de l'élément. La manipulation d'éléments comme les serveurs et les VM à partir d'un nom est utilisé par les langages des hyperviseurs et différents gestionnaires de VM comme virsh ou encore VMware vSphere. Par exemple, l'hyperviseur Xen propose le shell xm fonctionnant sur ce principe. Les VM présentes sur un hyperviseur sont listées à partir de la commande « list » :

```
> xm list
Name                                ID   Mem VCPUs      State   Time(s)
Domain-0                             0   389     1     r-----   1414.9
vm1                                      305     1     r-----    349.9
adminVm                                  300     1                 0.0
guestVm                              6   300     1     -p----     10.6
```

Le nom de chaque VM est alors précisé ainsi que différentes informations comme l'état du cycle de vie de la VM. Par exemple, la VM « vm1 » est démarrée (état : running). Pour démarrer la VM « adminVm », l'administrateur doit exécuter la commande :

```
xm start adminVm
```

L'utilisation des noms comme identifiant est donc naturelle pour l'administrateur. Dans le langage proposé, le démarrage de la VM précédemment énoncé s'écrit :

```
adminVm:start
```

Les éléments du modèle comme les serveurs et les VM ont un nom clairement identifié qui est utilisé lors des communications entre serveurs et entre hyperviseurs et VM. D'autres éléments comme les sites ou les baies hébergeant les serveurs n'ont pas de noms. L'administrateur doit donc leur en affecter un lors de la description de l'infrastructure. Ces noms doivent être choisis avec soin afin de faciliter la manipulation d'ensemble d'éléments.

4.2.1.2 Construction d'ensembles d'éléments

Afin d'exécuter une introspection ou une reconfiguration sur plusieurs éléments, la construction d'un ensemble d'élément est nécessaire. Dans ce but, l'opérateur « [] » permet la sélection de plusieurs éléments d'après leur nom.

Cet opérateur permet d'exprimer des énumérations afin de sélectionner plusieurs éléments ayant des noms proches. Par exemple, les éléments « serveur1 », « serveur2 », « serveur3 » et « serveur4 » sont sélectionnés à partir de l'expression suivante :

```
serveur[1-4]
> serveur1, serveur2, serveur3, serveur4
```

La réponse de la commande affiche les éléments de l'infrastructure sélectionnés. Si aucun élément ne correspond aux noms donnés, la réponse est vide. Un deuxième type d'énumération est possible afin de sélectionner les éléments « serveur.lille1 », « serveur.lille2 », « serveur.lille3 » :

```
serveur.[lille1, lille2, lille3]
< serveur.lille1, serveur.lille2, serveur.lille3
```

Les deux types d'énumération sont utilisables dans la même expression :

```
serveur.[nantes, lille, bordeaux][1-3]
```

La sélection de tous les éléments ayant une partie de leur nom commune est effectuée à l'aide du caractère « * ». Par exemple, la sélection de tous les éléments contenant la séquence de caractère « lille » est effectuée avec l'expression suivante :

```
*lille*
> serveur.lille1, serveur.lille2, serveur.lille3, nfs.lille
```

Grâce à ces expressions, l'administrateur peut rapidement construire un ensemble d'éléments pour effectuer une tâche d'administration. Cependant, la construction d'un ensemble d'éléments ne doit pas reposer uniquement sur des règles de nommages. La construction d'ensembles à partir d'éléments ayant des noms complètement différents doit aussi être disponible pour ne pas restreindre les opérations d'administration.

4.2.1.3 Opérations sur les ensembles

Pour les éléments possédant des noms radicalement différents, l'opérateur « [] » propose la construction d'ensembles à partir des opérations ensemblistes traditionnelles. Toutes les expressions ci-dessous sélectionnent uniquement les serveurs « serveur1 », « serveur2 », « serveur3 » et « serveur4 » :

1. sélection par l'union de deux ensembles :

   ```
   [ serveur[1-3], service4 ]
   ```

2. sélection par la différence de deux ensembles :

   ```
   [ serveur[1-5] ! serveur5 ]
   ```

3. sélection par l'intersection de deux ensembles :

   ```
   [ serveur[1-5] & serveur[1-4] ]
   ```

La sélection des éléments à partir de leur nom permet à l'administrateur de construire rapidement des ensembles afin de les manipuler. Cependant, ces sélections ne prennent pas en compte la hiérarchie du modèle et empêchent l'administrateur de sélectionner rapidement, par exemple, tous les fils d'un élément.

4.2.1.4 Navigation dans le modèle

Le type et la hiérarchie des éléments peuvent aussi être utilisés dans la sélection d'éléments. En effet, chaque élément se voit attribuer un type lors de la description de l'infrastructure. Ces types servent à créer une hiérarchie cohérente entre les éléments. Chaque relation père/fils est alors typée. À partir de cette hiérarchie typée, il est alors possible de récupérer tous les fils d'un élément grâce à l'opérateur « / ». Cette opération est appelée une navigation. Prenons la hiérarchie de types suivante : ville -> centre -> baie -> serveur. Un élément de type « ville » contient donc des éléments de type « centre » représentant des centres de données qui sont composés de baies contenant, à leur tour, des serveurs.

Comme les éléments de type « ville » sont les premiers éléments, ils n'ont pas de père. La sélection de tous les éléments de type « ville » se fait alors en omettant le premier paramètre de l'opérateur « / » :

```
/ville
> nantes, lille
```

Lorsqu'un père existe, l'obtention de tous ses fils est possible en utilisant une navigation. Par exemple, l'obtention des fils de type « centre » du site « nantes » est alors effectuée par la navigation suivante :

```
nantes/centre
> emn-center, polytech-center
```

La réponse permet de savoir qu'il existe deux centres de données localisés à Nantes. Cette navigation est dite « directe » car le type « centre » suit le type « ville » dans la hiérarchie. La navigation de type « indirect » est aussi autorisée. Les types intermédiaires sont alors omis. Par exemple, la liste de tous les serveurs du centre de données nommé « emn-center » est obtenue par la navigation :

```
emn-center/serveur
> kvm1, kvm2, kvm3, kvm4
```

Les noms des serveurs sont ainsi obtenus et peuvent être utilisés pour une navigation. Dans les exemples précédents, les navigations débutent d'un élément père pour atteindre des éléments fils. Les navigations étant bidirectionnelles, il est possible de connaître le site hébergeant le serveur « kvm1 » avec la commande suivante :

```
kvm1/site
> nantes
```

Lors d'une navigation fils/père, un seul élément sera obtenu car si un père peut avoir plusieurs fils, un fils ne peut avoir qu'un seul père.

La navigation est aussi utilisée pour parcourir les ponts et donc, naviguer d'une vue à une autre. Ajoutons à notre exemple, une deuxième vue contenant des VM et possédant la hiérarchie suivante : utilisateur -> vm. Un pont entre les types « serveur » et « vm » est créé pour symboliser la relation d'hébergement entre un hyperviseur et ses VM. À l'aide d'une navigation, on peut alors lister toutes les VM présentent sur un serveur :

```
kvm1/vm
> vm1, vm4
```

Une navigation indirecte permet d'obtenir tous les serveurs utilisés par l'utilisateur « invite1 » :

```
invite1/serveur
> kvm2, kvm3
```

La création d'un ensemble d'éléments peut utiliser les opérateurs « [] » et « / » dans la même expression :

```
[ nantes/serveur & invite[1-2]/serveur ]
```

À l'aide des opérateurs de constructions d'ensembles, l'administrateur peut sélectionner des ensembles importants d'éléments. Ces sélections vont ensuite être utilisées pour obtenir des informations sur l'infrastructure et réorganiser les ressources.

4.2.2 Introspection de l'infrastructure

4.2.2.1 Gestion de propriétés

Lors de l'administration d'une infrastructure virtualisée, les prises de décisions doivent prendre en compte l'organisation de l'intégralité des ressources. L'administrateur a donc besoin d'outils lui permettant de récupérer rapidement des informations sur la structure et sur les éléments de l'infrastructure. La récupération de ces informations est appelée « introspection de l'infrastructure ».

Les informations sur la structure permettent à l'administrateur d'établir des relations entre différents éléments de l'infrastructure. Par exemple, la relation d'hébergement entre les serveurs et les VM est représentée par une relation père/fils. L'administrateur voulant connaître les VM hébergées par le serveur « serveur1 » utilisera l'opérateur de navigation « / » précédemment introduit :

```
serveur1/vm
> vm1, vm2, vm3
```

Le même opérateur sera utilisé afin d'obtenir le serveur hébergeant la VM « vm1 » :

```
vm1/serveur
> serveur1
```

Pour obtenir des informations sur plusieurs éléments, l'administrateur peut construire un ensemble d'éléments comme vu précédemment.

Les informations sur les éléments aident l'administrateur dans ces prises de décisions, par exemple, pour réorganiser les ressources. Ces informations sont appelées les « propriétés des éléments ». Ces propriétés ajoutent de l'information sur un élément afin de faciliter son identification et de préciser la quantité d'une ressource produite ou consommée. Par exemple, la propriété « ip », disponible pour un serveur ou une VM, simplifie l'identification de l'élément en lui associant une adresse réseau. La propriété « mem » permet de définir la quantité de mémoire des éléments. Pour un serveur, cette propriété indique la quantité de mémoire offerte. Pour une VM, la propriété « mem » définit la quantité mémoire que la VM consomme sur le serveur ainsi que la mémoire offerte aux applications exécutées par la VM. Plusieurs propriétés sont prédéfinies pour simplifier l'administration des serveurs et des VM :

Les propriétés décrites ci-dessus permettent de préciser les ressources des serveurs et des VM. Deux types de ressources existent : les ressources statiques et les ressources dynamiques.

Les ressources statiques sont les ressources qui n'évoluent que rarement : cpu, cpu_nb, mem, power_min, power_max, mac. Ces ressources sont définies par la configuration du serveur et de la VM. Pour les modifier, il faut donc changer les composants d'un serveur ou reconfigurer les ressources affectées à la VM lors de la phase d'initialisation. Ces opérations étant coûteuses, elles restent occasionnelles. On peut donc considérer ces ressources comme étant statiques. La valeur de ces propriétés est définie lors de l'instanciation du modèle.

Les ressources dynamiques sont les ressources qui évoluent rapidement dans le temps : cpu_cons, cpu_free, mem_cons, mem_free, power_cons. Ces propriétés représentent l'évolution de la consommation des ressources. Par exemple, la consommation CPU d'une VM dépend des applications qu'elle exécute. La consommation CPU évoluera donc rapidement et influencera la consommation du serveur qui héberge la VM. L'actualisation de la valeur de ces propriétés ne peut donc pas être prise en charge par l'administrateur. Des outils seront donc chargés de mesurer

Nom de la propriété	Définition
cpu	la capacité d'un processeur
cpu_nb	le nombre de processeurs (pour les serveurs) ou de VCPU (pour les VM)
cpu_cons	la consommation cpu en pourcentage
cpu_free	la quantité de cpu disponible en MHz
mem	la capacité mémoire
mem_cons	la consommation mémoire en pourcentage
mem_free	la quantité de mémoire disponible en Mo
power_cons	la consommation électrique
power_min	la consommation électrique minimale d'un serveur
power_max	la consommation électrique maximale d'un serveur
mac	l'adresse mac
ip	l'adresse ip

TABLE 4.1 – Propriétés des éléments de type « serveur » et « VM »

différentes ressources et d'exporter les mesures vers le modèle. Ce dernier sert alors d'interface à l'administrateur qui obtient alors un accès à de multiples informations via une même interface.

Les consommations mémoire et CPU sont exprimées en pourcentage afin de permettre à l'administrateur de connaître rapidement l'état d'une VM ou d'un serveur (surchargé ou inactif). En effet, l'expression de la consommation CPU d'un serveur en MHz oblige l'administrateur à consulter deux autres propriétés : la capacité d'un CPU et le nombre de CPU du serveur. L'administrateur doit ensuite en déduire la quantité exacte de CPU disponible sur un serveur. Dans la plupart des cas, l'administrateur ne cherche pas à connaître la quantité de CPU restante mais plutôt l'utilisation du serveur dans l'infrastructure. Par exemple, un serveur ayant une consommation CPU de 10 % peut plus facilement être éteint qu'un serveur consommant plus et donc étant plus utilisé. De plus, dans une grande infrastructure, les serveurs ont des caractéristiques différentes notamment en termes de capacité mémoire et CPU. L'affichage des consommations CPU et mémoire respectivement en MHz et en Mo nécessiterait la consultation des capacités du serveur. Par exemple, une consommation mémoire de 2000 Mo n'a pas la même signification si le serveur possède une capacité mémoire de 2000 Mo ou de 4000 Mo.

Les consommations CPU et mémoire des VM correspondent aux ressources du serveur consommées par la VM. Si une VM possède 512 Mo de mémoire et est hébergée sur un serveur ayant 1024 Mo de mémoire alors la consommation mémoire de la VM sera de 50%. De cette façon, la somme des consommations mémoire ou CPU de toutes les VM hébergées par un serveur est égale à la consommation CPU ou mémoire du serveur. Par conséquent, si les trois VM d'un serveur consomment 10%, 20% et 30% de la ressource CPU de leur hôte, la consommation CPU du serveur sera de 60%.

La consultation d'une propriété par l'administrateur est effectuée avec l'opérateur « : ». Par exemple, la consommation CPU du serveur nommé « serveur1 » s'écrit :

```
serveur1:cpu_cons
```

La consultation d'une propriété n'existant pas ou n'ayant pas de valeur renvoie la valeur « empty ».

Afin de garantir la flexibilité du modèle, les propriétés peuvent être ajoutées et modifiées par l'administrateur après la construction du modèle. L'ajout de propriétés permet de prendre en

compte l'évolution de l'infrastructure. Par exemple, la différenciation entre des serveurs dotés d'un hyperviseur et des serveurs dédiés au stockage de fichiers pourra être faite par l'ajout d'une propriété « role » :

```
serveur[1-3]:role = nfs
serveur[4-40]:role = hyperviseur
```

Toutes les propriétés du modèle peuvent aussi être modifiées en utilisant la même syntaxe qu'un ajout de propriété. Par exemple, si un hyperviseur est installé sur le serveur nommé « serveur3 », la propriété « role » est modifiée de la façon suivante :

```
serveur3:role = hyperviseur
```

À partir de ces propriétés, l'administrateur peut donc obtenir des informations sur l'infrastructure et ainsi assurer une administration cohérente.

4.2.2.2 Filtres et sélections

Les propriétés associées aux éléments peuvent être utilisées pour effectuer une sélection plus précise d'un ou plusieurs éléments. En effet, des filtres peuvent être utilisés pour sélectionner un ou plusieurs éléments parmi un ensemble d'éléments préalablement sélectionnés. Par exemple, la sélection de tous les serveurs ayant un hyperviseur est définie grâce à l'opérateur « » de la façon suivante :

```
\serveur{role == hyperviseur}
```

Des fonctions peuvent aussi être utilisées afin d'obtenir une sélection de plusieurs éléments possédant des propriétés communes. Par exemple, la sélection des serveurs ayant la plus forte et la plus faible consommation CPU s'écrit respectivement :

```
\serveur{max(cpu_cons,1)}
\serveur{min(cpu_cons,1)}
```

Le deuxième paramètre des fonctions « max » et « min » représente le nombre d'éléments à sélectionner. Par exemple, la sélection des trois VM ayant le plus de mémoire est précisée par l'expression :

```
\vm{max(mem,3)}
```

La création de filtres basés sur la structure de l'infrastructure sont aussi disponible. De cette manière, l'administrateur peut sélectionner tous les serveurs hébergeant au moins une VM :

```
\serveur{size(vm) > 1}
```

L'utilisation des opérateurs « min » et « max » peuvent aussi être utilisés pour effectuer des sélections prenant en compte l'organisation des ressources. Par exemple, la sélection des serveurs hébergeant le plus et le moins de VM sont obtenus respectivement avec les expressions suivantes :

```
\serveur{max(vm,1)}
\serveur{min(vm,1)}
```

L'utilisation des propriétés permet à l'administrateur d'associer des informations à chaque élément de l'infrastructure. Ces informations sont utilisées pour affiner le résultat d'une sélection en restreignant le nombre d'éléments à administrer à l'aide de filtres. Le second rôle de ces propriétés est le suivi des ressources consommées comme le CPU ou la mémoire. L'administrateur peut alors surveiller l'état des ressources de l'infrastructure et mettre en œuvre des reconfigurations dans le but de maintenir l'équilibre entre les ressources fournies et les ressources demandées.

4.2.3 La reconfiguration de l'infrastructure

4.2.3.1 Gestion des serveurs et des machines virtuelles

Le but de l'administration est de maintenir une infrastructure dans un état cohérent assurant le bon fonctionnement des applications s'exécutant sur les serveurs. Une quantité de ressources insuffisante sur un serveur par rapport aux demandes des applications peut entraîner des dysfonctionnements des applications. La modification de l'organisation des ressources par l'administrateur passe par l'exécution de reconfigurations.

Dans une infrastructure virtualisée, la couche de virtualisation regroupe les applications dans des VM. Les besoins en ressources d'une VM correspond alors aux besoins de toutes les applications qu'elle héberge. Les hyperviseurs vont alors offrir à l'administrateur différentes reconfigurations permettant de gérer les VM.

La première reconfiguration est utilisée par l'administrateur pour la création de ressources, c'est-à-dire, la création de VM. La création d'une VM est la dernière étape de la phase d'initialisation. Cette phase lie l'image disque à la VM et configure le nombre de VCPU et la capacité mémoire de cette dernière. La création de l'image disque d'une nouvelle VM dans VMScript est une copie d'une image disque modèle préalablement créée. La création d'une image disque étant complexe, notamment avec l'installation du système d'exploitation, ce processus n'est pas pris en compte dans l'outil d'administration proposé ici. La création de multiples VM sur un serveur est possible en fournissant plusieurs noms à l'opérateur « createvm » :

```
serveur1:createvm(vm[1-3],lenny)
```

Afin de simplifier la syntaxe de l'opération de création de VM, les informations nécessaires à l'initialisation de la VM sont décrites dans un modèle de VM nommé « template ». Ce modèle regroupe le nombre de VCPU, la mémoire et le modèle de l'image disque à copier. Ainsi, pour la création d'une VM, seul le nom de l'élément et le modèle de la VM sont à préciser.

La création de la VM ne démarre pas la VM et donc ne consomme pas de ressources mémoire ou CPU. Seule la création de la nouvelle image disque consomme de la ressource en diminuant l'espace de stockage utilisé. Cette opération permet d'enregistrer toutes les caractéristiques de la VM afin de permettre l'opération de démarrage de la VM :

```
vm1:start
```

Si aucun serveur n'est donné à l'opérateur « start », la VM est démarrée sur le serveur sur lequel elle a été crée. Dans le cas d'un espace de stockage partagé entre plusieurs serveurs, la VM peut être démarrée sur un de ces serveurs. Dans ce cas, un serveur peut être spécifié lors du démarrage :

```
vm1:start serveur2
```

Une fois la VM démarrée, plusieurs reconfigurations sont possibles (voir le cycle de vie de la VM sur la figure 3.1). Si l'administrateur a besoin de libérer des ressources, par exemple, pour régler un problème de surcharge, deux reconfigurations lui sont proposées : l'arrêt, la destruction et la mise en pause de la VM. Lors de l'arrêt de la VM, le système d'exploitation et toutes les applications de la VM sont arrêtés. Les données non sauvegardées et les calculs en cours d'exécution sont perdus. L'arrêt d'une ou plusieurs VM est ordonné à l'aide de l'opérateur « stop » :

```
vm[1-3]:stop
```

La destruction d'une VM permet d'arrêter une VM sans passer par l'arrêt du système d'exploitation. Cette opération est utile lorsque le système d'exploitation ne répond plus. Cependant, les données non sauvegardées sur le disque dur sont perdues lors de cette opération. L'opérateur « destroy » est donc à utiliser avec précaution :

```
vm1:destroy
```

La mise en pause d'une VM permet à l'administrateur de libérer les ressources attribuées à la VM sans perdre les calculs en cours d'exécution. La mémoire, contenant les informations nécessaires aux calculs, est copiée sur le disque dur du serveur. La libération des ressources CPU et mémoire se fait donc au détriment de la ressource disque du serveur. La mise en pause d'une VM est effectuée par l'opérateur « suspend » :

```
vm1:suspend
```

Une fois la VM mise en pause, la reprise des calculs est effectuée à l'aide de l'opérateur « resume » :

```
vm1:resume
```

Les opérateurs « start », « stop », « suspend » et « resume » sont aussi applicables sur des serveurs provoquant le démarrage, l'extinction, la mise en pause et la reprise d'un serveur. Le démarrage d'un serveur à partir d'une application est une opération complexe car le système d'exploitation n'étant pas en cours d'exécution, il ne peut être utilisé pour allumer le serveur. Le démarrage du serveur reste cependant possible en utilisant la carte de management du serveur ou en agissant sur l'alimentation du serveur par le biais des prises électriques. L'arrêt d'un serveur peut servir à économiser de l'électricité ou à effectuer une maintenance, par exemple, remplacer des pièces défectueuses. Dans un but d'économie d'énergie, une alternative est possible si le matériel précédemment cité n'est pas présent dans l'infrastructure du centre de données. En effet, la mise en pause (aussi appelée mise en veille ou Suspend To Ram, en anglais) d'un serveur diminue de façon significative la consommation du serveur car le processeur et le disque dur ne sont plus alimentés. La reprise de l'activité du serveur est alors faite via la carte réseau qui, étant toujours alimentée, sort le serveur de son état de pause.

La gestion des applications d'une VM ou d'un serveur en cours d'exécution peut être fait à l'aide de l'opérateur « exec » responsable d'exécuter des commandes sur la VM. L'administrateur peut alors lancer ou arrêter des applications comme un serveur d'application tomcat :

```
vm1:exec(/lib/tomcat/startup.sh)
```

Le redémarrage d'un serveur ou d'une VM est régulièrement nécessaire. En effet, au fur à mesure de son utilisation, la mémoire peut être encombrée par des données qui ne sont plus utilisées. Le redémarrage permet alors de nettoyer la mémoire et donc d'améliorer les performances du serveur ou de la VM. De plus, un système d'exploitation est régulièrement mise à jour afin de corriger ou d'améliorer certaines fonctionnalités. À la suite d'une mise à jour importante, un redémarrage du système d'exploitation peut être requis. Par exemple, après la mise à jour de l'hyperviseur, le redémarrage du serveur est effectué avec l'opérateur « reboot » :

```
serveur1:reboot
```

Lorsqu'une opération de maintenance doit être effectuée sur un serveur, le serveur doit être arrêté afin de remplacer ou d'ajouter un composant. Un serveur ne peut être éteint si il héberge des VM. En effet, les applications dispensées par les VM seraient alors arrêtées en même temps que le serveur. Les VM hébergées par ce serveur doivent donc être éteintes ou migrées vers d'autres serveurs. Si ces applications sont en cours d'utilisation, leur arrêt perturberait fortement les utilisateurs. La migration d'une VM va alors permettre de régler le problème de façon transparente pour les utilisateurs. Cette reconfiguration est effectuée à l'aide de l'opérateur « migrate » :

```
vm1:migrate serveur2
```

En cas de surcharge d'un serveur, les opérateurs « stop », « destroy » et « suspend » permettent de suspendre l'activité d'une VM afin de libérer des ressources sur le serveur l'hébergeant. Cependant, certaines applications ne peuvent être arrêtées afin d'assurer la haute disponibilité d'un service. Dans ce cas, l'administrateur peut décider de migrer une VM sur un serveur disposant de suffisamment de ressources disponibles. L'administrateur, à l'aide des outils d'introspection, va donc identifier un serveur capable de recevoir une des VM du serveur surchargé - dans notre cas, le serveur1 - et procéder à la migration :

```
serveur1/vm
> vm1, vm2, vm3
/serveur{cpu_cons < 50}
> serveur3, serveur7
vm1:migrate serveur3
```

Dans une infrastructure de taille conséquente, il peut être difficile à l'administrateur de résoudre plusieurs problèmes de surcharge survenant simultanément, par exemple, lors d'une hausse importante des consommations des VM. Dans ce cas, l'opérateur « solve » propose à l'administrateur l'utilisation d'algorithmes de placement calculant les reconfigurations nécessaires afin de résoudre les problèmes de surcharges :

```
/serveur:solve(checker)
```

Cet opérateur prend en paramètre le nom de l'algorithme à utiliser. Suivant l'algorithme sélectionné, l'administrateur pourra tenter de résoudre les problèmes de surcharge en favorisant une politique d'ordonnancement. Dans l'exemple précédent, l'algorithme « checker » permet de corriger les problèmes de surcharges en exécutant le moins de reconfigurations possibles. La mise en place d'une politique d'ordonnancement comme l'équilibrage de charge ou la consolidation devient compliquée pour une infrastructure possédant de nombreuses VM. Les algorithmes

« loadbalancing » et « consolidation » vont alors permettre à l'administrateur d'effectuer respectivement un équilibrage de charge et une consolidation sur l'ensemble ou une partie de l'infrastructure.

Toutes les reconfigurations manipulent des ensembles décrits par les mécanismes de sélection présentés précédemment. L'opérateur « : » suivi d'un nombre N permet de choisir N éléments dans l'ensemble sélectionné. Ainsi, la sélection d'un serveur parmi tous les serveurs nantais disposant de plus de 2 Go de mémoire s'écrit :

```
nantes/serveur{mem_free> 2000}:1
```

L'administrateur peut alors facilement éteindre tous les serveurs sans VM ou démarrer cinq VM supplémentaires sur le site « nantes » :

```
/serveur{size(vm) == 0}:stop
vm[1,4,6,9,10]:start nantes/serveur{mem_free> 2000}:1
```

À partir des reconfigurations précédemment décrites, l'administrateur organise les ressources de l'infrastructure en agissant sur le cycle de vie des serveurs et des VM. Dans une infrastructure virtualisée, l'ajout, la suppression et le déplacement de ressources sont amplement simplifiées par l'utilisation de la virtualisation.

4.2.3.2 Programmation de reconfigurations

L'administrateur doit adapter l'infrastructure en fonction des besoins des applications hébergées. Une infrastructure surdimensionnée par rapport aux besoins des applications se traduit par l'hébergement de plus de VM que nécessaire. Le nombre important de VM rend alors l'infrastructure plus difficile à administrer et diminue le taux d'exploitation des serveurs. L'infrastructure est donc moins rentable car de nombreuses ressources sont gaspillées. Par exemple, des serveurs et des VM inutilisés consomment, respectivement, de l'électricité et de la mémoire.

Le sous dimensionnement de l'infrastructure engendre un manque de ressources pour les applications. Dans ce cas, le nombre d'applications hébergées par les VM est trop important par rapport aux ressources fournies par les serveurs. Par conséquent, les serveurs ne disposent pas de suffisamment de ressources pour toutes les VM et certaines applications manquent de ressources. Leur performance est alors dégradée. De plus, le sous dimensionnement de l'infrastructure rend l'administration plus difficile car l'administrateur n'a plus de marge pour prévoir les hausses de consommations des VM ou migrer une VM sur un serveur ayant des ressources disponibles. L'administrateur va donc devoir surveiller les besoins des applications afin de les satisfaire par un dimensionnement correct de son infrastructure.

Dans certains cas, l'évolution des besoins des applications peut être prévisible. Par exemple, dans le cadre d'applications de commerce en ligne, les besoins seront plus importants en dehors des horaires de bureau. Au contraire, pour des applications utilisées dans un environnement de travail, au sein d'une entreprise par exemple, les besoins seront proportionnels au nombre d'employés présents. Dans ces deux exemples, l'administrateur connaît les périodes de basse activité. Il peut donc décider de réduire le nombre de serveurs allumés pendant cette période et d'utiliser tous les serveurs pour répondre aux besoins des applications quand l'activité devient importante. L'administrateur peut donc écrire deux scripts : un script de regroupement de VM sur quelques serveurs et un script de déploiement de VM sur tous les serveurs. En considérant une infrastructure d'une dizaine de serveurs hébergeant cinquante VM, le script permettant le regroupement des VM sur trois serveurs est le suivant :

```
vm[1-16]:migrate serveur1
vm[17-34]:migrate serveur2
vm[35-50]:migrate serveur3
server[4-10]:stop
```

Le script de déploiement démarre les serveurs précédemment éteints et migre les VM afin d'utiliser tous les serveurs :

```
server{state == off}:start
vm[16-20]:migrate serveur4
vm[21-25]:migrate serveur5
vm[26-30]:migrate serveur6
vm[31-35]:migrate serveur7
vm[36-40]:migrate serveur8
vm[41-45]:migrate serveur9
vm[46-50]:migrate serveur10
```

Une fois ces scripts écrits, l'administrateur peut les appeler afin d'exécuter le regroupement ou le déploiement des VM :

```
eval(regroupement)
eval(déploiement)
```

À partir de ces scripts, l'administrateur économisera donc l'électricité de sept serveurs pendant la période de basse consommation de l'infrastructure. Cependant, l'administrateur devra exécuter ces scripts tous les jours à heure fixe afin de garantir l'économie d'énergie. De plus, si l'administrateur oublie de déployer les VM avant la hausse d'activité, les applications auront un fonctionnement dégradées jusqu'à la résolution du problème.

Afin d'éviter ces situations, l'outil d'administration propose la programmation de reconfigurations. Cette programmation est utilisée afin d'exécuter une tâche d'administration pendant une période définie ou de différer l'exécution d'une reconfiguration, par exemple, afin d'assurer une reconfiguration sans intervention de l'administrateur. La date de l'exécution est alors exprimée en utilisant la syntaxe de l'application crontab. Cet outil est utilisé par les administrateurs des systèmes d'exploitation Linux pour exécuter une tâche à un horaire précis et éventuellement périodique. Chaque programmation utilise la syntaxe suivante [Libb] :

```
mm hh jj MM JJ ma_tâche
mm: les minutes de l'horaire
hh: l'heure de l'horaire
jj: le jour de l'horaire
MM: le mois de l'horaire
JJ: le jour de la semaine (par exemple, 5 désigne le vendredi)
Chaque champs peut utiliser les notations suivantes :
* : chaque unité (0, 1, 2, 3,..., 8, 9) ;
5,8 : les unités 5 et 8 ;
2-5 : les unités de 2 à 5 (2, 3, 4, 5) ;
*/3 : toutes les 3 unités (0, 3, 6) ;
10-20/3 : toutes les 3 unité entre 10 et 20 (10, 13, 16, 19).
```

Dans le langage dédié aux reconfigurations, l'année a été ajoutée à l'horaire des tâches d'administration programmées et le caractère « : » est utilisé comme séparateur de champs. Ainsi, l'exécution d'une migration à 23 heures 30 tous les jours de l'année 2012 s'écrit :

```
[30:23:*:*:*:2012] vm1:migrate serveur1
```

La mise à jour applicative des VM à 3 heures puis leur redémarrage à 6 heures tous les deux jours et seulement les jours ouvrés s'exprime à l'aide de l'expression suivante :

```
[00:3:*/2:*:1-5:*] \vm:exec(apt-get update; apt-get dist-upgrade)
[00:6:*/2:*:1-5:*] \vm:reboot
```

Grâce à la programmation des reconfigurations, l'administrateur désirant économiser de l'électricité en dehors des horaires de l'entreprise pourra donc programmer l'exécution de ces scripts à l'aide des expressions suivantes :

```
[00:6:*:*:*:*] eval(déploiement)
[00:23:*:*:*:*] eval(regroupement)
```

À partir des reconfigurations, l'administrateur est capable de modifier l'organisation des ressources immédiatement ou de différer ces modifications afin de répondre à des besoins futurs. Cependant, plus l'infrastructure est de taille conséquente, plus les liens entre VM et entre VM et serveurs sont compliqués. Par exemple, plusieurs VM communicant fréquemment entre elles doivent rester sur le même serveur afin de garantir des performances optimales. D'autre part, une VM effectuant des calculs complexes devra être affectée au serveur ayant le plus de ressource CPU afin de diminuer ces temps de calcul. L'administrateur ne peut faire face à cette complexité sans des outils l'aidant à décrire et maintenir de telles règles d'administration.

4.3 Un langage pour le maintien de la cohérence de l'infrastructure

Les mécanismes de sélection, décrits dans la section 4.2.1, forment une base essentielle aux mécanismes d'introspection (voir section 4.2.2) et de reconfiguration (voir section 4.2.3). À l'aide de ces mécanismes, l'administrateur d'infrastructures de taille conséquente observe et organise des ressources virtualisées en manipulant principalement des serveurs et leurs VM associées. Cependant, avec la complexification des applications (calcul haute performance, applications N-tiers, etc.), des besoins spécifiques agissant sur l'environnement d'exécution des applications sont apparus. Par exemple, la haute disponibilité d'une application peut être requise pour une application de commerce en ligne où l'indisponibilité de l'application entraîne des pertes financières. Afin de garantir des taux de disponibilité de 99,9999 % (soit 31 secondes d'indisponibilité par an), l'utilisation de sauvegardes et de clones de VM peuvent être utilisés.

L'utilisation de sauvegardes de VM consiste à effectuer, à intervalles réguliers, une copie sur un espace de stockage de l'image disque et de la mémoire de la VM. Il est alors possible de reconstituer la VM et les calculs qui étaient en cours d'exécution. La perte de données est alors dépendante du temps écoulé entre la perte de la VM et la dernière sauvegarde. De plus, le temps d'interruption du service dépend de la taille de la sauvegarde servant à reconstituer la VM.

L'utilisation de clones de VM consiste à maintenir une copie de la VM originale en cours d'exécution. En cas de perte de la VM, les utilisateurs sont alors redirigés vers le clone. Dès que

la panne est détectée, le clone prend donc le relais. Le clone étant en cours d'exécution, aucun temps de démarrage ou d'initialisation du service n'est nécessaire. Cependant, cette technique est coûteuse en ressources. De plus, le choix des serveurs hébergeant la VM et son clone est important afin d'assurer une haute disponibilité. Par exemple, si la VM et son clone sont hébergés par le même serveur, une panne survenant sur le serveur arrête le service. La haute disponibilité de l'application n'est donc plus assurée. Une règle de placement doit donc définir la relation entre la VM et son clone afin de maintenir les deux VM sur des serveurs distincts. Lors de l'organisation ou de la réorganisation des ressources de l'infrastructure, l'administrateur doit donc prendre en compte cette règle de placement enfin de garantir la haute disponibilité de l'application.

4.3.1 Relations entre les VM et les serveurs

Sur une infrastructure de quelques serveurs, l'administrateur peut effectuer les reconfigurations nécessaires au bon fonctionnement des applications en vérifiant les règles de placement manuellement. Cependant, dès que la taille de l'infrastructure augmente, la sélection et la vérification des règles de placement concernant tel serveur ou telle VM se compliquent rapidement. Les règles de placement doivent alors être intégrées dans l'outil d'administration afin d'aider l'administrateur à maintenir une organisation cohérente des ressources de son infrastructure. Deux types de règles de placement existent : les règles entre une VM et un serveur et les règles entre VM.

Les ressources attribuées à une VM dépend de son initialisation mais aussi du serveur hébergeant la VM. Par exemple, de nombreux modèles de processeurs sont disponibles sur le marché et certains sont (beaucoup) plus performants que d'autres. Lors de la phase d'initialisation de la VM, l'administrateur fixe le nombre de processeur attribuée à la VM mais les performances de ces processeurs sont définies par le type des processeurs installés sur le serveur hébergeant la VM. De plus, les processeurs évoluant rapidement, il est courant d'avoir dans une infrastructure des processeurs différents suivant la date de conception du serveur. En règle générale, la phase d'initialisation de la VM permet de fixer la quantité de ressources attribuée à la VM et l'affectation de la VM à un serveur définit la qualité de la ressource. Par exemple, la carte graphique utilisée par la VM étant celle du serveur, l'administrateur préférera héberger une VM dédiée aux traitements d'images numériques sur un serveur ayant une carte graphique puissante. Il est donc naturel pour un administrateur de vouloir réglementer l'hébergement de VM ayant des besoins spécifiques. La définition d'une relation entre un ensemble de VM et un ensemble de serveurs utilise la syntaxe suivante :

```
identifiant_de_la_regle: VM opérateur serveurs
```

L'opérateur définit la relation entre les VM et les serveurs sélectionnés. La sélection des éléments est effectuée à partir des mécanismes de sélection présentés dans la section 4.2.1. L'identifiant est utilisé pour modifier la règle de placement.

La complexification des applications et les besoins de l'informatique dans les nuages ont entrainé la création de relations entre les applications et donc entre les VM. Pour des besoins de haute disponibilité, de travail collaboratif, de sécurité, etc., des relations entre les VM ont été définies. Par exemple, la haute disponibilité nécessite la séparation d'une VM et de son clone. Les performances d'applications travaillant ensembles croissent avec l'amélioration des communications réseau. L'administrateur souhaite alors regrouper les VM hébergeant ce type d'applications sur un seul serveur. Pour des raisons de sécurité, un utilisateur, exécutant des applications critiques, voudra isoler ces VM des autres VM de l'infrastructure. Un ou plusieurs serveurs seront alors dédiés à l'hébergement des VM de cet utilisateur. Des règles de placement

entre VM sont donc nécessaires. Contrairement aux règles précédentes, l'affectation des VM aux serveurs n'est pas spécifiée. L'administrateur ne sait donc pas sur quels serveurs les VM vont s'exécuter. La relation définit uniquement l'interaction entre les VM. Par exemple, lors de la séparation d'une VM et de son clone, les serveurs hébergeant la VM et le clone ne sont pas précisés. L'administrateur souhaite simplement que les deux VM ne soient pas hébergées par le même serveur. La définition d'une relation entre plusieurs VM utilise la syntaxe suivante :

```
identifiant_de_la_regle: VM opérateur
```

L'opérateur définit la relation entre les VM sélectionnées. La sélection des VM est effectuée à partir des mécanismes de sélection présentés dans la section 4.2.1.

À partir de règles de placement, l'administrateur peut spécifier des relations entre VM et entre VM et serveurs. Ces règles doivent alors être respectées afin de maintenir une organisation cohérente des ressources de l'infrastructure.

4.3.2 Les règles de placement

Dans une infrastructure virtualisée, les règles de placement permettent à l'administrateur de définir des besoins spécifiques régissant l'organisation des ressources. Ces besoins sont dictés par plusieurs problématiques comme la sécurité, la configuration réseau, etc. L'outil d'administration proposé définit quatre opérateurs de placement aidant l'administrateur à définir les relations entre les VM et les serveurs de son infrastructure.

L'opérateur « on » définit des règles de placement établissant une relation d'hébergement entre un ensemble de VM et un ensemble de serveurs. Chaque VM intervenant dans cette relation doit alors être hébergée par un des serveurs sélectionnés. Par exemple, la règle suivante contraint les VM « vm1 », « vm2 » et « vm3 » à être hébergées par les serveurs « serveur1 » ou « serveur2 » :

```
power+: vm[1-3] on serveur[1,2]
```

Toutes les VM sélectionnées ne sont pas nécessairement hébergées par le même serveur. Cette règle va permettre, par exemple, d'attribuer les serveurs les plus puissants à quelques VM nécessitant une forte puissance de calcul. Dans certaines infrastructures composées de grappe de serveurs, les communications réseau entre grappes sont impossibles ou doivent être limitées à cause de liaisons réseaux de faible capacité. Dans ce cas, l'administrateur peut partitionner la totalité des VM à administrer sur les grappes. Les migrations entre les grappes sont alors interdites. L'exemple suivant montre le partitionnement de cinq cents VM sur une infrastructure contenant cent serveurs :

```
cl1: vm[1-100] on serveur[1-20]
cl2: vm[101-200] on serveur[21-40]
cl3: vm[201-300] on serveur[41-60]
cl4: vm[301-400] on serveur[61-80]
cl5: vm[401-500] on serveur[81-100]
```

L'opérateur « noton » définit des règles de placement établissant une relation d'exclusion entre un ensemble de VM et un ensemble de serveurs. Les VM intervenant dans la relation ne doivent pas être hébergées par un des serveurs sélectionnés. Cet opérateur est utilisé dans le même contexte que l'opérateur « on ». Il permet cependant de faciliter l'écriture d'une règle « on » portant sur la quasi totalité des serveurs de l'infrastructure. Par exemple, la règle « on » suivante concerne 98 des 100 serveurs disponibles dans l'infrastructure :

```
on13: vm[1-3] on [ serveur[1-35], serveur[37-56], serveur[58-100] ]
```

La règle précédente affecte 3 VM à 98 serveurs. Le même résultat serait obtenu en utilisant la relation d'exclusion suivante :

```
noton13: vm[1-3] noton serveur[36,57]
```

La syntaxe est alors plus simple à écrire mais aussi à lire et le risque d'erreur est plus faible.

Les règles « on » et « noton » vont restreindre le nombre de serveurs sur lesquels un ensemble de VM sera hébergé. Ces règles augmentent donc la complexité de l'administration car certaines VM auront des comportements différents des autres. Par exemple, en cas de surcharge d'un serveur, les VM ne pourront pas être migrées sur n'importe quel serveur. De plus, des règles trop restrictives peuvent être impossibles à vérifier. Par exemple, l'hébergement de toutes les VM par seulement 2 serveurs de l'infrastructure ne sera pas réalisable car les VM manqueront cruellement de ressources :

```
allvm: vm[1-500] on serveur[1-2]
```

L'opérateur « group » définit des règles de placement établissant une relation de cohabitation entre les VM sélectionnées. Toutes les VM de l'ensemble intervenant dans la règle doivent être voisine, c'est-à-dire, hébergées par le même serveur. Dans le cas d'applications N-tiers, les VM contenant ces applications communiqueront régulièrement. Le regroupement de ces VM sur un même serveur permet de diminuer la ressource réseau utilisée tout en améliorant les communications entre les applications et donc d'accroitre leur performance. La règle suivante définit les VM « vm1 », « vm2 », « vm3 » et « vm4 » comme voisines :

```
voisine: vm[1-4] group
```

La création de cohabitations de VM complexifie l'administration en créant des groupes de ressources pouvant être difficiles à manipuler. En effet, la migration d'une des VM de la cohabitation entraine une migration de toute la cohabitation et donc d'un groupe de ressources. L'administrateur doit alors s'assurer que les ressources disponibles sur le serveur sont suffisantes pour accueillir la totalité des VM de la cohabitation.

L'opérateur « spread » définit des règles de placement établissant une relation d'éloignement entre les VM sélectionnées. Les VM de l'ensemble intervenant dans la règle ne doivent pas être voisines. Dans un contexte de haute disponibilité, une VM et son clone ne doivent pas être hébergés par un même serveur afin d'assurer la disponibilité du service en cas de panne sur un serveur. La relation d'éloignement entre la VM « vm2 » et son clone est définit avec la syntaxe suivante :

```
eloignement: [ vm2, vm2-clone ] spread
```

Dans une relation d'éloignement, la cardinalité de l'ensemble de VM définit le nombre de serveurs nécessaires afin de respecter la règle de placement. Plus le groupe de VM à éloigner est important, plus les possibilités de migrations d'une VM de ce groupe seront réduites.

Les règles de placement ajoutent des relations entre les serveurs et les VM servant à mettre en place des besoins spécifiques d'administration comme la haute disponibilité. Cependant, des règles trop restrictives peuvent gêner l'administration de l'infrastructure en limitant l'exécution des reconfigurations.

Une règle de placement représente un besoin spécifique définit par l'administrateur. Cependant, la mise en place de règles de placement n'engendre pas de reconfiguration. Lorsqu'une règle de placement doit être corrigée, les reconfigurations permettant la correction de cette règle sont multiples. Le choix des reconfigurations à exécuter est alors à la charge de l'administrateur. Lorsqu'une règle n'est pas respectée, elle est signalée à l'administrateur qui choisit les reconfigurations à exécuter pour corriger la règle. Si la règle n'est plus d'actualité, l'administrateur peut la supprimer ou la désactiver.

4.3.3 Gestion des règles de placement

4.3.3.1 Manipulation des règles de placement

L'organisation des ressources évolue en fonction des applications hébergées par l'infrastructure et de leur consommation. Les règles de placement, afin de s'adapter à de nouveaux besoins, doivent pouvoir être modifiées.

La détection des règles de placement non respectées alerte l'administrateur en cas de problèmes concernant l'organisation des ressources. L'administrateur reçoit alors un message indiquant la règle posant problème :

```
eloignement: [ vm2, vm2-clone ] spread
```

Si la règle de placement non respectée n'est plus utile, l'administrateur la supprime à l'aide de l'opérateur « del » :

```
eloignement:del
```

Si la règle de placement ne peut être respectée pendant une certaine période, par exemple, à cause de la panne d'un serveur, l'administrateur peut désactiver cette règle avec l'opérateur « disable » :

```
eloignement:disable
```

Une fois le problème corrigé, l'activation de la règle est effectuée par l'opérateur « enable » :

```
eloignement:enable
```

Afin de vérifier les règles s'appliquant à une ou plusieurs VM, l'administrateur peut lister les règles à partir de l'opérateur « rules » :

```
vm[1-3]:rules
> eloignement: [ vm2, vm2-clone ] spread [ON]
> cl1: vm[1-100] on serveur[1-20] [OFF]
```

Dans l'exemple précédent, deux règles s'appliquent sur les VM « vm1 », « vm2 » et « vm3 ». La règle nommée « eloignement » concerne les VM « vm2 » et « vm2-clone ». Elle apparaît avec la même syntaxe que celle utilisée lors de sa création afin de permettre à l'administrateur de connaître son impact sur les autres éléments de l'infrastructure et de pouvoir la modifier facilement. La règle est actuellement active d'où la présence du mot clé « ON ». La seconde règle concerne la partitionnement des VM sur vingt serveurs. Cette règle est actuellement désactivée. Elle ne sera donc pas vérifiée par l'outil d'administration.

Certains besoins d'administration sont en rapport avec une activité périodique comme la création d'un projet d'entreprise. L'entreprise en question loue un serveur et 3 VM afin d'utiliser une infrastructure virtualisée pour héberger les applications liées au projet. Dans ce cas, une règle de placement peut être ajoutée avant la création du projet par l'administrateur. Afin de ne pas perturber l'organisation des ressources avant le début du projet, la règle sera activée qu'à la date de commencement du projet, le 20 novembre :

```
[00:8:20:11:12:*] projetX: vm[1-3] on serveur1
```

Si la fin du projet est connue, la suppression de la règle peut aussi être programmée :

```
[00:8:20:11:12:* - 00:20:20:23:12:*] projetX: vm[1-3] on serveur1
```

À partir des mécanismes de gestion des règles de placement, l'administrateur peut maintenir à jour les règles de placement correspondant aux besoins de l'infrastructure. Cependant, si de nombreuses règles ne sont pas respectées, le calcul des reconfigurations à exécuter afin de réparer la totalité des règles peut être complexe.

4.3.3.2 Résolution autonomes des règles de placement

L'outil d'administration avertit l'administrateur des règles de placement à corriger. Cependant, le maintien et la correction des règles additionnés à la mise en place d'une politique d'ordonnancement comme l'équilibrage de charge ou la consolidation peut s'avérer ardu pour des infrastructures de taille conséquente. L'opérateur « solve » (voir section 4.2.3) peut alors calculer les reconfigurations à exécuter afin de mettre en place la politique d'ordonnancement demandée en prenant en compte les règles de placement.

L'administrateur peut alors appliquer une politique d'ordonnancement sur un ensemble de serveurs. Cet ensemble de serveurs définit les règles de placement à prendre en compte lors du calcul des reconfigurations résolvant les problèmes détectés. Prenons l'exemple d'une infrastructure contenant 4 serveurs et 3 VM. Les trois VM sont réparties sur trois des quatre serveurs de sorte que la VM « vm_i » soit hébergée par le serveur « serveur_i ». Les règles de placement suivantes ont été définies :

```
vm[1-3] group
vm[1,2] on serveur[1-3]
vm3 on serveur4
```

L'administrateur souhaite alors résoudre les règles de placement relatives à deux serveurs :

```
serveur[1-2]:solve(checker)
```

La politique de vérification des règles commence par sélectionner toutes les règles agissant sur les serveurs sélectionnés :

```
vm[1-3] group
vm[1,2] on serveur[1-3]
```

La première règle est sélectionnée car les VM « vm1 » et « vm2 » sont hébergées par les serveurs « serveur1 » et « serveur2 ». La deuxième règle se réfère directement aux serveurs sélectionnés, elle est donc aussi traitée. La résolution du problème est donc effectuée sur 2 serveurs, 2 VM et 2 règles.

La règle de cohabitation (« group ») doit être modifiée afin de supprimer la VM « vm3 » des éléments à traiter. Si cette VM n'est pas supprimée, le serveur hébergeant la VM « vm3 », c'est-à-dire le serveur « serveur3 » doit être inclut à l'ensemble des éléments à analyser. Les VM qu'il héberge doivent aussi être ajoutées au problème ainsi que les règles s'appliquant à ces éléments. La taille du problème a donc augmenté. De plus, l'ajout de la VM « vm3 » ajoute une règle de placement supplémentaire. Cette règle ajoute à son tour le serveur « serveur4 » au problème.

Afin d'éviter l'accroissement du problème à résoudre définit par l'administrateur, les règles partiellement inclues sont modifiées pour ne conserver seulement les éléments sélectionnés par l'administrateur.

Cependant, la suppression des éléments dans une règle de placement peut entraîner une résolution partielle de la règle. Dans l'exemple ci-dessus, la règle « vm[1-3] group » sera modifiée pour devenir : « vm[1-2] group ». La résolution de cette règle entraînera une résolution partielle car seules les VM « vm1 » et « vm2 » seront regroupées.

La sélection des ressources utilisées dans le calcul du plan de reconfiguration permet à l'administrateur de :

- résoudre des problèmes de tailles raisonnables. Suivant l'algorithme utilisé, les temps de résolution peuvent être importants. La définition de problèmes plus petits évite les longs temps de calcul gênant pour la cohérence du plan de reconfiguration. En effet, durant la phase de calcul, l'évolution des ressources n'est pas prise en compte et donc plus le temps de calcul est long, plus les risques d'erreur dans le plan de reconfiguration est important. Un exemple d'erreur serait une migration qui ne peut aboutir car il n'y a plus assez de ressources pour accueillir la VM ;
- appliquer des politiques d'ordonnancement différentes sur plusieurs parties de l'infrastructure. Par exemple, utiliser une politique d'équilibrage de charge sur un site et une politique de consolidation sur un autre :

```
nantes/serveur:solve(loadbalancing)
lyon/serveur:solve(checker)
```

À l'aide de l'opérateur « solve », l'administrateur calcule les reconfigurations nécessaires à la résolution des règles de placement. L'administrateur possède alors des outils lui permettant d'obtenir des informations sur l'organisation des ressources (introspection), de décrire des besoins spécifiques à l'aide de relations entre les éléments (règle de placement) et de modifier l'organisation des ressources (reconfiguration) afin de maintenir la cohérence de son infrastructure.

4.4 Conclusion

L'utilisation de langages dédiés permet à l'administrateur de manipuler des ensembles d'éléments facilitant les opérations sur de nombreux éléments. À partir de deux langages dédiés, une approche langage est proposée dans le but de fournir à l'administrateur les outils nécessaires pour l'administration d'infrastructures virtualisées. Le premier langage permet une description complète de l'infrastructure avec diverses vues, chaque vue représentant une problématique comme la virtualisation ou l'électricité. À l'aide du second langage, l'administrateur observe l'organisation des ressources et la modifie en fonction des besoins des applications. Des règles de placement sont aussi proposées pour définir des relations entre VM et entre VM et serveurs afin de maintenir une organisation cohérente des ressources.

Chapitre 5

VMScript : un outil d'administration d'infrastructures virtualisées

L'outil d'administration proposé s'intéresse à la gestion d'infrastructures virtualisées de taille conséquente. Pour administrer un grand nombre de serveurs et de VM, l'outil assiste l'administrateur en analysant l'organisation des ressources afin de détecter les serveurs surchargés et les règles de placement non respectées. De plus, la cohérence des reconfigurations demandées est vérifiée en simulant la reconfiguration sur le modèle. L'outil est alors capable d'empêcher les reconfigurations provoquant une incohérence comme la violation d'une règle de placement. En cas de détection d'incohérences, l'administrateur dispose de deux méthodes pour réorganiser les ressources de l'infrastructure. La première, basée sur la programmation par contraintes, cherche à résoudre toutes les incohérences au risque de ne pas trouver la solution parfaite. La seconde, basée sur la programmation génétique, essaie d'améliorer l'organisation actuelle des ressources sans garantir une organisation optimale. Les infrastructures contenant plusieurs centaines de serveurs et de VM obligent l'outil d'administration à être flexibles afin de s'adapter à la taille de l'infrastructure mais aussi à son hétérogénéité. L'outil d'administration est basé sur une architecture distribuée permettant l'utilisation de la puissance de calcul de plusieurs serveurs garantissant des temps d'exécution corrects. L'architecture modulaire de l'infrastructure est utilisée pour adapter l'outil aux infrastructures hétérogènes.

Sommaire

Après avoir détaillé la première contribution de cette thèse dans le chapitre précédent, à savoir le langage d'administration d'infrastructures virtualisées de taille conséquente, nous nous intéressons dans ce chapitre à la deuxième contribution : l'environnement d'exécution du langage. Après une description succincte en section 5.1 des principes architecturaux, les mécanismes permettant de garantir la cohérence des règles d'administration seront détaillés en section 5.2. En section 5.3, l'architecture distribuée mise en place sera décrite ainsi que l'approche modulaire retenue et les mécanismes de distribution utilisés. Cette approche distribuée apporte une grande flexibilité en permettant l'ajout, le retrait et le remplacement de certains composants durant l'exécution, mais également de permettre le passage à l'échelle de la solution. Ce point particulier est traité en section 5.5.

5.1 Principes architecturaux

L'outil d'administration proposé reprend l'architecture MAPE-K [Inc06] utilisée habituellement par les ordonnanceurs de VM autonomes (voir figure 5.1). L'observation de l'infrastructure est réalisée par deux modules : le module d'observation et le module de surveillance. Le module d'observation remonte les ressources consommées par les serveurs et les VM. Le module de surveillance détecte, quant à lui, les problèmes d'organisation de ressources comme une consommation excessive de ressource CPU ou une règle de placement non respectée. Le module d'exécution communique avec les serveurs afin de modifier l'organisation des ressources. La partie d'analyse et de planification des reconfigurations est effectuée par l'administrateur à l'aide du langage dédié ou par le module de placement. Ce module calcule un plan de reconfiguration réorganisant les ressources de l'infrastructure en fonction des besoins décrits par l'administrateur, par exemple, une politique d'ordonnancement de consolidation. La base de connaissance regroupant toutes les règles de placement définies dans le système est alors nécessaire. L'intervention de l'administrateur à diverses étapes de l'architecture MAPE-K fait de l'outil d'administration un système non autonome. En effet, l'administrateur agit sur les différents modules par l'intermédiaire du langage dédié. Par exemple, l'introspection est réalisée par interaction avec le module d'observation et l'exécution de reconfigurations passe par l'utilisation du module d'exécution. L'administrateur est donc le seul maître de son infrastructure en étant à l'origine de chaque reconfiguration.

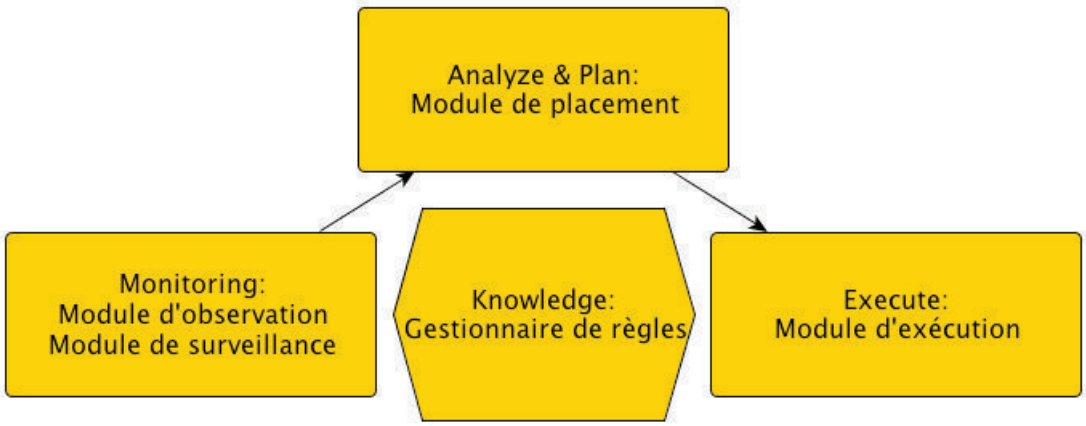

Figure 5.1 – Architecture MAPE-K utilisé par VMScript

5.2 Maintien de la cohérence de l'infrastructure

Un des principaux buts de l'administrateur est de maintenir une organisation cohérente des ressources de l'infrastructure virtualisée. La cohérence de l'organisation est dictée par les ressources demandées par les VM, le respect des règles de placement et la mise en place d'une politique d'ordonnancement. Afin d'aider l'administrateur à organiser correctement les VM qu'il administre, quatre modules vont intervenir. Le gestionnaire de règles vérifie la cohérence entre les règles définies par l'administrateur. Le module d'exécution garantit que les reconfigurations n'introduisent pas d'incohérence dans l'infrastructure. Le module de surveillance détecte les problèmes présents sur l'infrastructure et le module de placement aide l'administrateur à résoudre ces problèmes et à mettre en place une politique de placement.

5.2.1 Cohérence des règles de placement

L'ajout, la modification et la suppression des règles de placement sont gérés par le gestionnaire de règles. Ce module assure également la cohérence de la base de règles en vérifiant, à chaque ajout de règles, que la nouvelle règle n'introduit pas une contradiction.

En effet, l'insertion de multiples règles peut engendrer une base de règles contenant des contradictions, c'est-à-dire un ensemble de règles ne pouvant être vérifié quelque soit l'organisation de l'infrastructure. Afin d'éviter les contradictions, le gestionnaire de règles vérifie, à chaque insertion, que la base de règles est cohérente. Si une contradiction est détectée, la règle n'est pas insérée et l'administrateur est averti des règles provoquant la contradiction. L'exemple ci-dessous montre une contradiction apparaissant dans une infrastructure comprend trois serveurs et deux VM.

```
r1: vm[1,2] group
r2: vm1 noton serveur[1-2]
r3: vm2 noton serveur3
```

La première règle maintient les VM sur le même serveur. La deuxième règle empêche la VM « vm1 » d'être hébergée par les serveurs « serveur1 » et « serveur2 ». La troisième règle empêche la VM « vm2 » de s'exécuter sur le serveur « serveur3 ». Comme l'infrastructure comprend trois serveurs, le groupe formé par les VM ne peut s'exécuter sur aucun serveur. Aucune organisation de l'infrastructure permet de respecter toutes les règles, il y a donc une contradiction.

Trois types de contradiction sont détectées. La première contradiction pouvant être détectée est l'appartenance de deux VM à une règle « group » et à une règle « spread ». L'exemple suivant montre une contradiction engendrée par des règles « group » et « spread » :

```
r1: vm[1,2] group
r2: vm[1,3] group
r3: vm[2,3] spread
```

Les VM « vm1 » et « vm2 » doivent être sur le même serveur. Comme les VM « vm1 » et « vm3 » doivent aussi être sur le même serveur, alors les VM « vm2 » et « vm3 » doivent être sur le même serveur. Du coup, la règle « spread » entraîne une contradiction car elle ne pourra jamais être vérifiée sans violer une autre règle.

Le deuxième type de contradiction détecté apparaît quand deux VM d'une règle « spread » doivent être hébergé par un même serveur. Un exemple de ce type de contradiction est illustré par l'exemple suivant :

```
r1: vm[1-4] spread
r2: vm[1-4] on serveur[1,2]
```

La première règle force les quatre VM à être hébergées par quatre serveurs. La deuxième règle engendre une contradiction en forçant les quatre VM à s'exécuter sur seulement deux serveurs.

La dernière contradiction est détectée lorsqu'une VM ou un groupe de VM ne peut être hébergée par aucun serveur de l'infrastructure. Deux cas de figure provoquent cette contradiction. Le premier est la présence d'un groupe de VM nécessitant plus de ressources statiques (par exemple, la mémoire) qu'en offre le plus gros serveur. Le deuxième cas est la présence d'un trop grand nombre de règles « on » ou « noton » qui empêchent une VM de s'exécuter sur tous les serveurs de l'infrastructure.

Lors de l'ajout d'une règle programmée, le gestionnaire de règles sélectionne l'ensemble des règles actives pendant la période d'activation de la règle à insérer puis vérifie la cohérence de la base de règles au moment de l'activation de la règle.

5.2.2 Cohérence des reconfigurations

Les reconfigurations de l'infrastructure décidées par l'administrateur visent à modifier l'organisation de l'infrastructure dans le but d'en améliorer son fonctionnement. Par exemple, l'administrateur migre toutes les VM d'un serveur afin de pouvoir l'éteindre ou le mettre en maintenance (par exemple, pour changer des composants défaillants). Ces reconfigurations doivent donc être cohérentes avec l'organisation actuelle des ressources et des règles de placement ajoutées par l'administrateur. Dans le but d'éviter des reconfigurations incohérentes, le module d'exécution effectue plusieurs tests afin de vérifier les conséquences de la reconfiguration sur l'infrastructure. Si la reconfiguration ajoute un problème aux problèmes déjà existant sur l'infrastructure, la reconfiguration est annulée, c'est-à-dire, qu'elle n'est pas transmise au serveur concerné. Dans ce cas, un message d'erreur est envoyé à l'administrateur. Les différents tests effectués avant chaque reconfiguration sont autant de préconditions devant être vérifiées par chaque reconfiguration. Ces préconditions portent sur le cycle de vie de la VM, la modification des ressources des serveurs et le respect des règles de placement actives.

La figure 5.2 définit le cycle de vie d'une VM. Ce diagramme définit les reconfigurations accessibles sur une VM pour un état de son cycle de vie précis. Par exemple, une VM dans l'état « suspended » doit subir la reconfiguration « resume » avant de pouvoir accéder à l'état « off » correspondant à une VM éteinte. Le module d'exécution utilise ce diagramme pour vérifier que l'état actuel de la VM correspond à la reconfiguration demandée. Les préconditions sont vérifiées au moment où le module d'exécution ordonnance la reconfiguration et non au moment où l'administrateur envoie la reconfiguration au modèle d'exécution. Le fonctionnement du module d'exécution pour la transmission d'une reconfiguration à l'hyperviseur peut être résumé par l'algorithme suivant :

```
1 - réception des reconfigurations envoyées par l'administrateur
2 - ordonnancement des reconfigurations en fonction des acteurs
3 - vérification des préconditions
4 - envoi des reconfigurations à l'hyperviseur concerné
```

La vérification des préconditions intervient juste avant la transmission de la reconfiguration à l'hyperviseur. En effectuant le test le plus tard possible, l'administrateur peut demander une reconfiguration « resume » et une reconfiguration « stop » sans attendre la fin de la première reconfiguration et donc la mise à jour de son cycle de vie. Si la précondition était vérifiée à

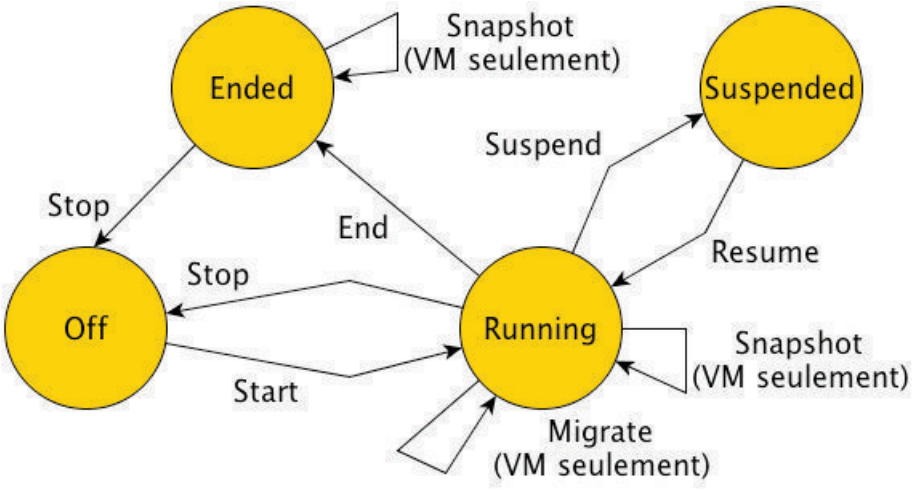

FIGURE 5.2 – Le cycle de vie d'un serveur ou d'une machine virtuelle. Les flèches montrent les reconfigurations modifiant l'état de la VM ou du serveur. L'état « ended » permet de spécifier que le serveur ou la VM doit être arrêté. Pour un serveur, cet état permet de programmer une maintenance. Le serveur doit être vidé de ses VM et ne peut pas accueillir de VM supplémentaires. Pour une VM, l'état « ended » permet à l'administrateur d'effectuer une sauvegarde avant l'extinction de la VM.

la réception de la reconfiguration, l'administrateur devrait attendre la fin de la reconfiguration « resume » avant de pouvoir demander la reconfiguration « stop ».

Les reconfigurations sont utilisées afin d'améliorer l'organisation des ressources de l'infrastructure. Cependant, une reconfiguration peut aussi détériorer cette organisation. Par exemple, la migration d'une VM vers un serveur n'ayant plus de ressource CPU disponible entraîne un manque de ressources sur toutes les VM du serveur. Si une telle migration se produit, la VM migrée voit son fonctionnement perturbé et perturbe aussi le fonctionnement des VM voisines. Afin d'éviter à l'administrateur d'ajouter une erreur en voulant améliorer l'organisation de l'infrastructure, les préconditions vérifient que les quantités de ressource CPU et mémoire sont suffisantes pour exécuter la reconfiguration. Trois reconfigurations possèdent ce type de précondition : le démarrage d'une VM, la sortie de pause d'une VM et la migration. En effet, ces reconfigurations augmentent les consommations CPU et mémoire d'un serveur. Il est donc indispensable de vérifier que le serveur dispose d'assez de ressources pour exécuter la reconfiguration. En revanche, les reconfigurations, comme l'arrêt ou la mise en pause d'une VM, supprimant des ressources à un serveur ne sont pas assujetties à ce type de préconditions.

La dernière précondition à être vérifiée est le respect des règles de placement. Une règle de placement est un besoin spécifique qui a été décrit par l'administrateur afin de garantir certaines propriétés. Une reconfiguration ne doit donc pas violer une règle qui était respectée avant son exécution. La vérification des règles de placement est divisée en deux étapes : la sélection des règles à vérifier et leur vérification.

Lorsqu'une reconfiguration est demandée, toutes les règles agissant sur un des acteurs de cette reconfiguration sont sélectionnées. Ensuite, les règles inactives et les règles non respectées sont retirées de la sélection. Cette dernière contient donc des règles respectées dans l'organisation actuelle de l'infrastructure. La reconfiguration est ensuite simulée sur le modèle. Par exemple, une migration est exécutée sur le modèle en modifiant le serveur hébergeant la VM. Les règles préalablement sélectionnées sont alors vérifiées à partir du nouveau modèle. Si une des règles

n'est plus respectée, la reconfiguration est annulée.

5.2.3 Détection de incohérences sur l'infrastructure

Deux types de problèmes peuvent rendre une infrastructure incohérente : les serveurs surchargés et les règles de placement non respectées. Les serveurs surchargés sont des serveurs disposant d'une quantité de ressource CPU insuffisant par rapport aux besoins des VM qu'il héberge. Lorsqu'un serveur est surchargé, les applications et les services s'exécutant sur la VM sont dégradés. L'hyperviseur a aussi besoin de la ressource CPU. En cas de surcharge, son fonctionnement sera dégradé et la gestion des VM qu'il héberge en sera impactée. Par exemple, le temps nécessaire pour effectuer une migration augmentera. Si la surcharge est importante, le serveur peut tomber en panne. Les VM sont alors brutalement arrêtées et leur image disque peut être corrompue.

Le deuxième type de problème est la violation d'une règle de placement. Une règle définit un placement précis d'une ou plusieurs VM. Lorsqu'une règle n'est pas respectée, une ou plusieurs VM n'ont pas un placement valide. Ce problème peut entraîner des pertes de performances et/ou des pertes de fonctionnalités au sein de l'infrastructure. Par exemple, deux VM hébergeant les mêmes services sont maintenus sur des serveurs différents par une règle de placement. Si cette règle n'est pas respectée et que le serveur hébergeant les VM tombe en panne, les services rendus par les VM ne sont plus disponibles.

Sur une infrastructure de taille conséquente, l'administrateur ne peut surveiller en permanence la consommation CPU des serveurs et l'état de toutes les règles. Le module de surveillance effectue donc cette tâche périodiquement et avertit l'administrateur des éventuels problèmes détectés. Le module de surveillance effectue seulement une observation des ressources de l'infrastructure. Ce module n'exécute jamais de reconfigurations. L'administrateur étant averti des problèmes présents sur l'infrastructure, il est libre de corriger les problèmes comme il l'entend. De plus, la consommation CPU d'une VM peut être soumise à de fortes et rapides variations. La surcharge d'un serveur peut donc être temporaire. L'administrateur peut donc décider de ne pas corriger un problème. Le module de surveillance est incapable de prédire les ressources consommées par une VM, il ne peut donc connaître la durée de la surcharge d'un serveur. Si de nombreux problèmes sont détectés, le calcul des reconfigurations à exécuter pour maintenir la cohérence de l'infrastructure est complexe. Dans ce cas, l'administrateur peut faire appel au module de placement chargé de l'aider dans cette tâche.

5.2.4 Le module de placement

Le module de placement est capable de mettre en place une politique d'ordonnancement dans l'infrastructure. Ce module peut être appelé périodiquement ou par l'administrateur. VMScript n'étant pas un système autonome, les appels périodiques sont désactivés par défaut. Trois politiques d'ordonnancement sont actuellement utilisées par ce module : la cohérence, l'équilibrage de charge et la consolidation.

La politique de cohérence corrige les problèmes tels que la surcharge de serveurs et le non respect des règles. Grâce à cette politique, l'administrateur réorganise les ressources de l'infrastructure sans chercher à atteindre un objectif précis. Aucune reconfiguration non essentielle à la cohérence de l'infrastructure n'est donc générée.

La politique d'équilibrage de charge répartit la consommation CPU des VM équitablement sur les serveurs de l'infrastructure tout en respectant les règles de placement. Cette politique est utilisée pour prévenir les pics de charge importants. L'application d'une politique d'équilibrage de charge ou de consolidation est plus coûteux que l'application d'une politique de cohérence. En effet, ces deux politiques nécessitent des calculs supplémentaires pour, respectivement, partager

la charge entre les serveurs ou minimiser le nombre de serveurs hébergeant des VM. Le temps de calcul pour trouver la solution est donc plus important. De plus, le nombre de reconfigurations exécutées risque d'augmenter car, aux reconfigurations résolvant les incohérences, il faut ajouter celles répartissant la charge ou minimisant le nombre de serveurs sans corriger d'incohérences.

La politique de consolidation minimise le nombre de serveurs hébergeant des VM. Dans le but d'économiser de l'électricité, les serveurs sans VM sont éteints ou mis en pause afin de consommer très peu d'énergie et de faciliter la dissipation de la chaleur. La baisse de la température dans l'infrastructure réduit l'énergie consommée en diminuant l'utilisation des climatiseurs. Cependant, cette politique est seulement utilisée dans les infrastructures ne connaissant pas de pics de charge soudains. En effet, la consolidation utilise la quasi totalité des ressources des serveurs hébergeant des VM. Si un pic de charge survient, celui-ci sera supporté par seulement quelques serveurs de l'infrastructure. De plus, ces serveurs ayant peu de ressources disponibles, de nombreuses migrations vont avoir lieues. Afin d'exécuter ces migrations, les serveurs éteints devront être allumés, une opération coûteuse en temps. La réorganisation de l'infrastructure peut alors être longue et réduire les performances des applications.

Le choix de la méthode de résolution Les algorithmes d'ordonnancement sont des algorithmes complexes dont l'implémentation influence fortement les temps de calcul nécessaires à l'obtention d'un plan de reconfigurations. Les ressources d'une infrastructure étant dynamiques, le temps de calcul d'un plan de reconfigurations doit être court. Si le calcul est trop long, les ressources auront fortement variées et les reconfigurations ne correspondront plus aux besoins de l'infrastructure.

Deux types d'algorithmes de placement sont disponibles dans l'outil d'administration proposé : les algorithmes basés sur la programmation par contrainte (PPC) et les algorithmes basés sur la programmation génétique (PG). Les algorithmes basés sur la PPC résultent d'une intégration d'Entropy [HLM10] dans l'outil d'administration et ne seront pas détaillés ici. Ces algorithmes cherchent à résoudre tous les problèmes de l'infrastructure. Si aucune solution n'est trouvée, aucune reconfiguration n'a lieu. Ces algorithmes utilisent un solveur de contraintes pour le calcul de la solution ce qui les rend difficilement parallélisable. L'approche des algorithmes basés sur la PG diffère car le but est d'améliorer l'organisation des ressources de l'infrastructure sans forcément résoudre tous les problèmes détectés. Ces algorithmes sont basés sur une exploration aléatoire de l'ensemble des solutions potentielles et sont facilement parallélisables.

La programmation génétique Les algorithmes d'ordonnancement implémentés dans l'outil d'administration reprennent le fonctionnement d'algorithmes basés sur la PG déjà existant. Un individu représente une organisation de l'infrastructure. Il est composé de gènes symbolisant les relations d'hébergement entre les VM et les serveurs (voir figure 5.3).

La population initiale est construite à partir de deux individus : l'organisation courante et une organisation créée aléatoirement. La création du deuxième individu permet d'effectuer une reproduction lors de la première génération. Cette création a lieu en affectant une valeur à chaque gène, c'est-à-dire en répartissant les VM sur les différents serveurs de l'infrastructure, tout en veillant au respect des ressources mémoire.

```
1 // Parcours de tous les gènes
2 for (i = 0; i < vms.length; i++) {
3         values.clear();
4         for (int count = 0; count < config.getNbServer(); count++)
5                 values.add(count);
6         boolean insertVm = false;
```

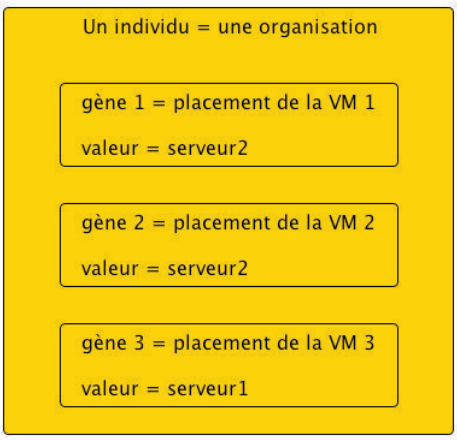

FIGURE 5.3 – Modélisation d'une organisation à l'aide d'un individu

```
7           // Insertion de la VM
8           while (!insertVm) {
9                   // Sélection aléatoire du serveur
10                  idx = rand.nextInt(values.size());
11                  iServer = values.get(idx);
12                  // Vérification de la ressource mémoire
13                  memServer[iServer] -= config.getMemVm(i);
14                  if (memServer[iServer] < 0) {
15                          // Retrait du serveur si mémoire insuffisante
16                          memServer[iServer] += config.getMemVm(i);
17                          values.remove(idx);
18                  } else
19                          insertVm = true;
20          }
21          // Affectation de la VM au serveur
22          newind.hosting(i, iServer);
23  }
```

Une fois la population initiale générée, plusieurs générations sont effectuées. Afin d'obtenir des résultats probants, l'algorithme génétique doit être étalonné afin de régler les différents paramètres. Le premier paramètre à définir est le nombre de générations. Ce paramètre est calculé en fonction du nombre de VM hébergées par l'infrastructure. En effet, plus il y a de VM, plus l'ordonnancement est complexe et donc un nombre plus important de générations est nécessaire. Chaque génération commence par une phase de reproduction. Le nombre de reproductions est exprimé en pourcentage de la taille de la population. Pour chaque reproduction, deux individus sont sélectionnés aléatoirement et un troisième individu est créé en inversant les gènes des deux individus.

```
1   Individual son1 = new Individual(config);
2   Individual son2 = new Individual(config);
3   int limit = vms.length / 2;
4   for (int i = 0; i < limit; i++) {
5           son1.vms[i] = this.vms[i];
6           son2.vms[i] = pInd.vms[i];
7   }
8   for (int i = limit; i < vms.length; i++) {
9           son1.vms[i] = pInd.vms[i];
10          son2.vms[i] = this.vms[i];
11  }
```

Les individus créés par reproduction sont ensuite analysés afin de corriger les serveurs hébergeant trop de VM par rapport à leur ressource mémoire. Les VM responsables des problème sont alors déplacées aléatoirement d'un serveur à un autre. Une fois corrigés, les individus sont ajoutés à la population et sont sujets à des mutations. Le nombre de mutations est défini par un pourcentage de la population. Chaque mutation provoque la modification aléatoire d'un ou plusieurs gènes. Le nombre de gènes modifiés lors d'une mutation est exprimé en pourcentage du nombre total de VM. Les mutations prennent en compte les ressources mémoire des VM et des serveurs. Les individus n'ont donc pas besoin d'être corrigés.

```
1  Individual mutant = new Individual(config);
2  Random rand = new Random();
3  // Ressources mémoire totales des serveurs
4  int[] tempMemServer = mutant.memServer.clone();
5  // Retrait de la mémoire consommée par les VM
6  for (int i = 0; i < mutant.vms.length; i++)
7          tempMemServer[mutant.vms[i]] -= mutant.config.getMemVm(i);
8  // Modification des gènes
9  for (int i = 0; i < pNbMutation; i++) {
10         // Sélection aléatoire de la VM à déplacer
11         int idVm = rand.nextInt(mutant.vms.length);
12         // Ressource mémoire de la VM
13         int myMem = mutant.config.getMemVm(idVm);
14         // Calcul des serveurs pouvant recevoir la VM
15         List<Integer> idPms = new ArrayList<Integer>();
16         for (int p = 0; p < mutant.config.getNbServer(); p++)
17                 if (tempMemServer[p] >= myMem && p != mutant.vms[idVm])
18                         idPms.add(p);
19         // Déplacement de la VM et mise à jour des ressources mémoire
20         tempMemServer[mutant.vms[idVm]] += myMem;
21         mutant.vms[idVm] = idPms.get(rand.nextInt(idPms.size()));
22         tempMemServer[mutant.vms[idVm]] -= myMem;
23 }
```

Les mutants sont ensuite eux aussi ajoutés à la population. La phase d'évaluation de la population peut commencer afin de réduire la taille de la population. La taille maximale de la population à conserver pour la prochaine génération est définie en fonction du nombre total de VM. Les individus de la population sont classés suivant trois critères :

- le nombre de problèmes (règles non respectées ou serveurs surchargés) détectés sur l'individu ;
- le respect de la politique d'ordonnancement ;
- le nombre de reconfigurations nécessaires (essentiellement des migrations) pour passer de l'organisation courante à l'organisation décrite par l'individu.

Dans l'algorithme implémenté, la priorité est donnée à la résolution des problèmes puis à l'application de la politique d'ordonnancement. Le coût de reconfiguration est le moins prioritaire. Ces priorités sont exprimées en triant les individus par ordre croissant du nombre de problèmes détectés. Les individus ayant le même classement sont ensuite triés par rapport au respect de la politique d'ordonnancement. Les individus ayant un classement identique sont départagés par rapport à leur coût de reconfiguration.

```
1  // Parcours de tous les individus
2  for (int i = 0; i < individuals.size() && !insert; i++)
3          // Classement lié au nombre de problèmes détectés
4          if (indi.problems() < individuals.get(i).problems()) {
5                  individuals.add(i, indi);
```

```
6                 insert = true;
7         } else if (indi.problems() == individuals.get(i).problems()) {
8                 // Prise en compte de la politique d'ordonnancement
9                 int indiPol = policy(indi);
10                int iPol = policy(individuals.get(i));
11                if (indiPol < iPol) {
12                        individuals.add(i, indi);
13                        insert = true;
14                } else if (indiPol == iPol)
15                        // Prise en compte du coût de la reconfiguration
16                        if (indi.planCost(init) < individuals.get(i).planCost(init)) {
17                                        individuals.add(i, indi);
18                                        insert = true;
19                        }
20        }
21
```

L'évaluation de la politique de consolidation est faite en comptant le nombre de serveurs hébergeant une ou plusieurs VM. Plus ce nombre est petit, meilleur est l'individu.

```
1  protected int policy(Individual pInd) {
2         Set<Integer> hosts = new HashSet<Integer>();
3         for (int i = 0; i < config.getVms().size(); i++)
4                 hosts.add(pInd.getHost(i));
5         return hosts.size();
6  }
```

L'évaluation de la politique d'équilibrage de charge est un peu plus complexe car elle passe par le calcul de l'écart type sur les consommations CPU des serveurs. Plus l'écart type est grand, plus les différences de consommation CPU entre serveurs sont grandes et donc moins l'individu est bon.

```
1  protected int policy(Individual pInd) {
2         int serverSize = config.getPms().size();
3         int[] load = new int[serverSize];
4         int average = 0;
5         for (int i = 0; i < config.getVms().size(); i++)
6                 load[pInd.getHost(i)] += config.getVms().get(i).getCpuCons();
7         for (int i = 0; i < serverSize; i++)
8                 average += load[i];
9         average /= serverSize;
10        int std = 0;
11        for (int i = 0; i < serverSize; i++)
12                std += (int) Math.pow(load[i] - average, 2);
13        return std;
14 }
```

Afin d'obtenir des résultats cohérents, l'algorithme génétique doit être calibré en étudiant l'impact de chaque variable. L'étalonnage de l'algorithme génétique est décrit dans la section **??** du chapitre dédié aux évaluations.

5.3 D'un cœur monolithique à un système distribué

En informatique, l'architecture logicielle décrit les interactions entre différents éléments d'une ou plusieurs applications. La première architecture ayant été utilisée est l'architecture monolithique non modulaire. C'est le cas des premières versions des systèmes d'exploitation Linux.

Dans ces systèmes, toutes les fonctionnalités sont regroupées dans une seule grosse application. Les systèmes monolithiques non modulaires ont une architecture logicielle très simple et une excellente vitesse d'exécution. Cependant, l'intégration de nouvelles fonctionnalités augmentent considérablement leur complexité et l'application devient alors difficile à maintenir et à faire évoluer. De plus, toutes les fonctionnalités sont obligatoirement chargées en mémoire qu'elle soit utilisée ou pas. Les applications modulaires sont alors apparues. Dans ces applications, une base non modulaire regroupant toutes les fonctionnalités fondamentales est créée et les autres fonctionnalités sont créées par l'ajout de modules. Comme les modules sont toujours intégrés à l'application, on parle d'architecture monolithique modulaire. Le côté monolithique de l'application permet de conserver de très bonnes vitesses d'exécution. De plus, l'application charge en mémoire seulement les modules qu'elle utilise. Cette architecture est utilisée dans le système d'exploitation Linux [Jon01]. Les modules sont utilisés pour le chargement de pilotes afin de s'adapter aux matériels utilisés et aussi pour le chargement de l'interface graphique, par exemple, pour charger une interface minimaliste au lieu d'une interface complète plus gourmande en ressources. L'architecture monolithique modulaire permet la conception d'applications ayant de bonnes performances sans perte de fonctionnalités. Cependant, comme tout système monolithique, si une erreur intervient dans un des modules, toute l'application est impactée. Afin de pallier ce problème, les modules doivent être sortis de l'application comme pour les systèmes d'exploitation à micro-noyaux [Lie95]. Dans de telles architectures, la base regroupant les fonctionnalités fondamentales est minimaliste et les autres fonctionnalités sont rendues par des modules externes à l'application. Un problème survenant dans un module ne peut plus corrompre la globalité de l'application. De plus, les possibilités de configurations augmentent car un module peut être remplacé par un autre module possédant les mêmes fonctionnalités sans reconfigurer toute l'application. Le principal inconvénient de ce type d'architecture est une perte de performance due à des communications fréquentes et coûteuses entre les modules et la base de l'application. De plus, les interfaces doivent être fixes car, en cas de modifications, tous les modules sont à modifier.

Dans l'outil d'administration proposé, l'utilisation des modules est nécessaire dans le but d'adapter l'outil d'administration à l'infrastructure gérée. Cette adaptation est effective en remplaçant un module par un autre module offrant les mêmes fonctionnalités. L'outil d'administration est composé de 10 modules (voir figure 5.4), chacun offrant une fonctionnalité d'administration.

L'ensemble des fonctionnalités de l'outil d'administration sont couvertes par les modules présentés dans la figure 5.4. La particularité de l'architecture de l'outil d'administration proposé est de pouvoir fonctionner soit en mode monolithique soit en mode distribué. Dans le premier cas, l'ensemble des modules, dits internes, sont chargés dans une même application Java. L'ensemble des fonctionnalités s'exécute à l'intérieur d'une même JVM (Java Virtual Machine, en anglais). Dans le second cas, certains modules, dits externes, peuvent être exécutés dans des processus différents et communiquent entre eux par des liaisons réseaux. La section 5.3.1 décrit à la fois les modules internes proposés dans l'outil d'administration et les mécanismes utilisés par l'approche distribuée.

5.3.1 Les modules internes pour un cœur monolithique

Les modules internes correspondent à l'utilisation d'une architecture monolithique de l'outil d'administration. Pour le chargement des modules internes, l'outil d'administration utilise le procédé de chargement de classes dynamiques du langage Java. Cette technique permet de charger en mémoire seulement les classes utilisées. Le choix des modules internes à charger est spécifié dans un fichier de configuration (voir l'annexe 8.4 pour plus de détail sur le fichier de confi-

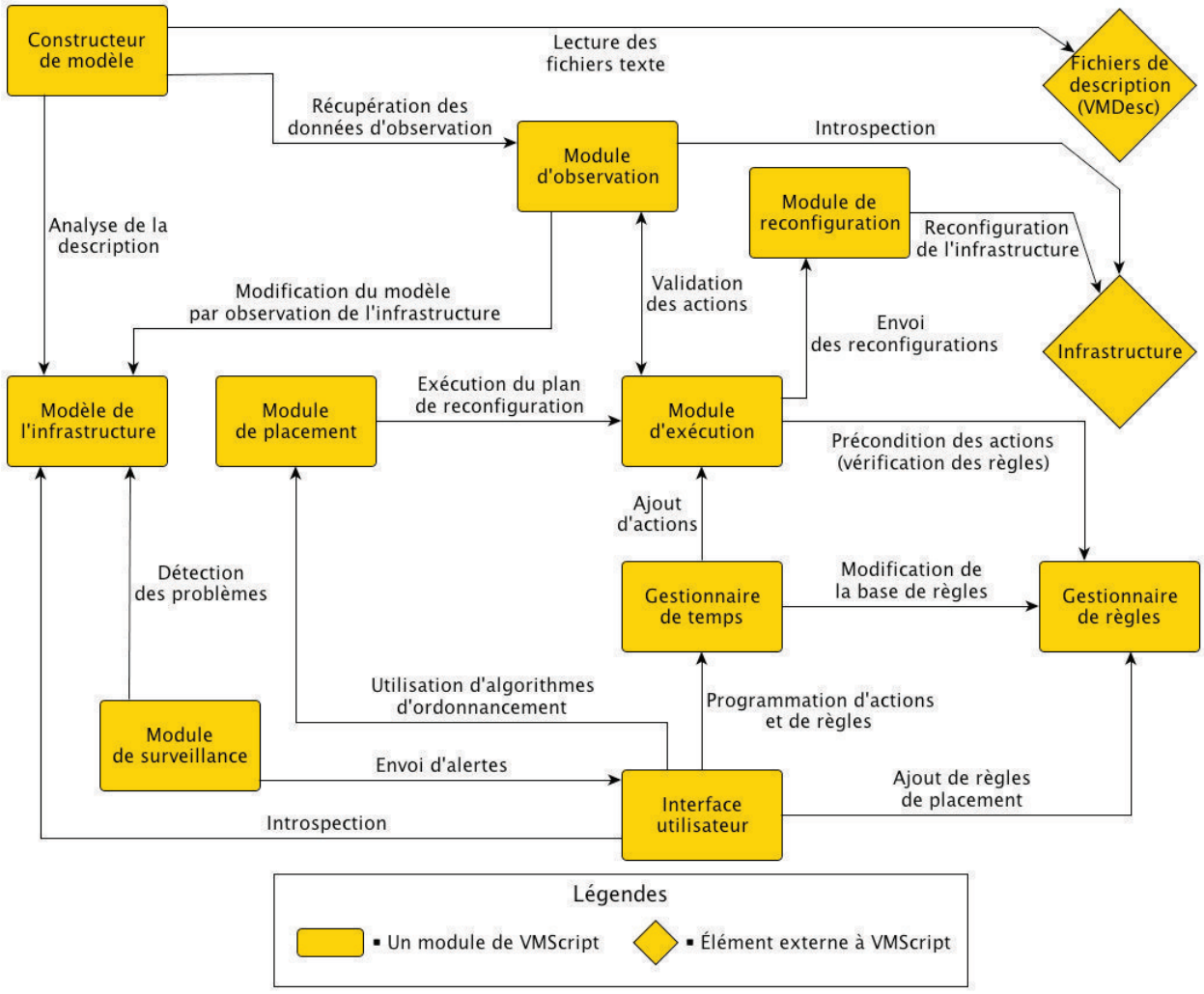

FIGURE 5.4 – Les différents modules de l'outil d'administration

guration). Le code suivant montre le chargement du module d'observation à partir de la classe « MonitoringKVM ». Ainsi, seul le module gérant l'hyperviseur KVM est chargé en mémoire. Les autres modules d'observation (par exemple, pour l'hyperviseur VMware ESXi) ne sont pas chargés et donc réduisent la consommation mémoire de l'outil d'administration.

```
1 Class<?> myClass = Class.forName("vmscript.backend.monitoringSystem");
2 Constructor<?> myConstructor = myClass.getConstructor(Integer.class);
3 myConstructor.newInstance(new Integer(3132));
```

Le module d'observation est responsable de l'analyse de l'organisation de l'infrastructure. Ce module doit détecter toutes les modifications dans l'organisation des ressources de l'infrastructure. L'analyse de l'infrastructure comprend donc la récupération des ressources consommées par tous les serveurs et toutes les VM, de leur cycle de vie et de l'emplacement des VM. L'utilisation d'un module d'observation est nécessaire afin de détecter les incidents survenant sur l'infrastructure comme l'arrêt brutal d'un serveur lié à une panne d'alimentation électrique. Ce module est aussi indispensable si plusieurs gestionnaires de VM sont utilisés sur l'infrastructure. Ainsi, les modifications apportées par un gestionnaire de VM externe à l'outil d'administration sont détectées et l'administrateur en est avisé. La récupération des ressources consommées offre la possibilité à l'administrateur de connaître la quantité de ressources consommées sur un serveur ou un VM. Par exemple, l'administrateur peut récupérer la ressource CPU consommée par une VM ou la mémoire libre disponible sur un serveur.

La récupération des consommations en ressources des VM est importante pour pouvoir réorganiser les VM de façon précise. Cependant, la récupération de la ressource CPU consommée par le biais de l'hyperviseur donne une valeur erronée lorsque l'hyperviseur se trouve surchargé, c'est-à-dire que toute sa ressource CPU est utilisée. Par exemple, si deux VM nécessitent 60 % de la ressource CPU de l'hyperviseur et qu'une troisième en nécessite 90 %, les consommations observées seront, par exemple, de 30 %, 30 % et 40 %. L'hyperviseur est donc bien surchargé mais l'administrateur ne sait pas exactement combien de VM il faut migrer. Par exemple, dans le cas précédent, la migration d'une VM n'est pas suffisante pour régler le problème de surcharge. De plus, comme l'administrateur ne connait pas exactement la consommation des VM, il ne sait pas quelle quantité de ressources prévoir pour satisfaire les besoins des VM. Il est donc difficile de choisir sur quel serveur la VM doit être migrée. Une solution à ce problème est l'installation d'une sonde mesurant la consommation CPU à l'intérieur de la VM. Cependant, cette solution est rarement mise en place car elle nécessite d'installer et de configurer une sonde à l'intérieur de chaque VM. Cette opération est longue lorsque les VM sont déjà existantes mais elle peut être moindre si la sonde est installée par défaut lors de la création des VM.

L'expérience suivante met en évidence le problème décrit ci-dessus. Trois outils sont utilisés pour la gestion de la ressource CPU d'un hyperviseur :

- un générateur de charge utilisé pour consommer une quantité précise de CPU. La consommation demandée est exprimée en pourcentage ;
- un script Bash permet d'observer la consommation CPU des VM. Ce script utilise l'outil dstat afin de mesurer facilement la consommation CPU ;
- un outil récupérant la consommation CPU à partir des données générées par l'hyperviseur ;

Le code source des outils utilisés est disponible à l'annexe 8.3.1. L'outil dstat est un utilitaire écrit en Python permettant de mesurer les ressources utilisées par des applications sur un système d'exploitation Linux. Les ressources observées sont nombreuses : CPU, disque, mémoire, interruptions matérielles, etc. De plus, de nombreux modules d'extensions sont utilisables pour des besoins d'observation plus spécifique, par exemple, pour relever la fréquence du processeur ou le pourcentage de charge de la batterie.

L'expérience consiste à générer une charge CPU sur des VM hébergées par un hyperviseur KVM. Les caractéristiques du serveur et des VM utilisées sont les suivantes :

- une capacité CPU de 2 GHz répartie sur deux processeurs (Intel(R) Core(TM)2 Duo CPU T5750) et 4 Go de RAM. Le serveur supporte la virtualisation matérielle ;
- l'hyperviseur utilisé est un KVM version 0.14.1 installé sur un système d'exploitation x86_64 Ubuntu GNU/Linux avec un noyau 3.0.0-14 ;
- les VM utilisent une image disque de 1 Go avec un VCPU et 600 Mo de RAM. Leur système d'exploitation est un x86_64 Debian GNU/Linux 6.0 avec un noyau 2.6.32-5-amd64 et sans interface graphique.

Afin de surcharger l'hyperviseur, six VM sont en cours d'exécution et consomment de la ressource CPU (voir figure 5.5). Le générateur de charge permet de configurer les VM de la façon suivante :

- deux VM consomment 20 % de leur CPU ;
- deux VM consomment 40 % de leur CPU ;
- deux VM consomment 70 % de leur CPU.

Dans la suite de l'expérience, l'observation de la charge CPU des VM est faite à partir de l'hyperviseur (à l'extérieur des VM) et à partir de l'outil dstat installé sur le système d'exploitation des VM (à l'intérieur des VM). Les résultats (voir figure 5.5) montrent des différences significatives entre les deux types d'observation.

Comme l'hyperviseur est surchargé, les VM ne disposent donc pas d'assez de ressources pour fournir la charge demandée par le générateur. Les valeurs de consommation CPU relevées à l'intérieur des VM sont donc inférieures aux valeurs attendues bien qu'il soit toujours possible de différencier les VM qui consomment 40 % de celles consommant 70 %. La surcharge du serveur est visible en faisant la somme de la consommation CPU des VM. Cette somme est égale à 200% car les deux CPU du serveur sont chargés à 100%. En revanche, lors de l'observation des charges CPU à partir de l'hyperviseur, les différences entre les charges CPU s'estompent et il devient impossible de différencier les VM consommant 40 % de celles consommant 70 %. De plus, la somme des consommations CPU est inférieure à 100%, le serveur n'est donc plus détecté comme surchargé. Le relevé détaillé des consommations CPU sont disponibles à l'annexe 8.3.2.

Le modèle est la représentation interne de l'infrastructure utilisée par l'outil d'administration afin d'agréger la description statique de l'infrastructure avec l'organisation et la consommation des ressources des différents éléments. La cohérence entre l'infrastructure et le modèle est conservée grâce au module d'observation qui reporte sur le modèle tous les changements qu'il détecte.

Le constructeur de modèle crée le modèle à partir des fichiers de description et/ou en se connectant au module d'observation.

Le module de reconfiguration est responsable d'exécuter les reconfigurations sur l'infrastructure afin de modifier l'organisation des ressources. Les modules de reconfiguration et d'observation servent de lien entre le modèle et l'infrastructure. Trois types de modules de reconfiguration et d'observation sont disponibles :

- les modules de type « dummy » permettent l'utilisation de l'outil d'administration sans le relier à une infrastructure. Toutes les reconfigurations sont exécutées directement sur le modèle et les ressources consommées n'évoluent pas ;
- les modules de type « vmflow » permettent de simuler une infrastructure en faisant évoluer les ressources consommées et le cycle de vie des VM et des serveurs ;

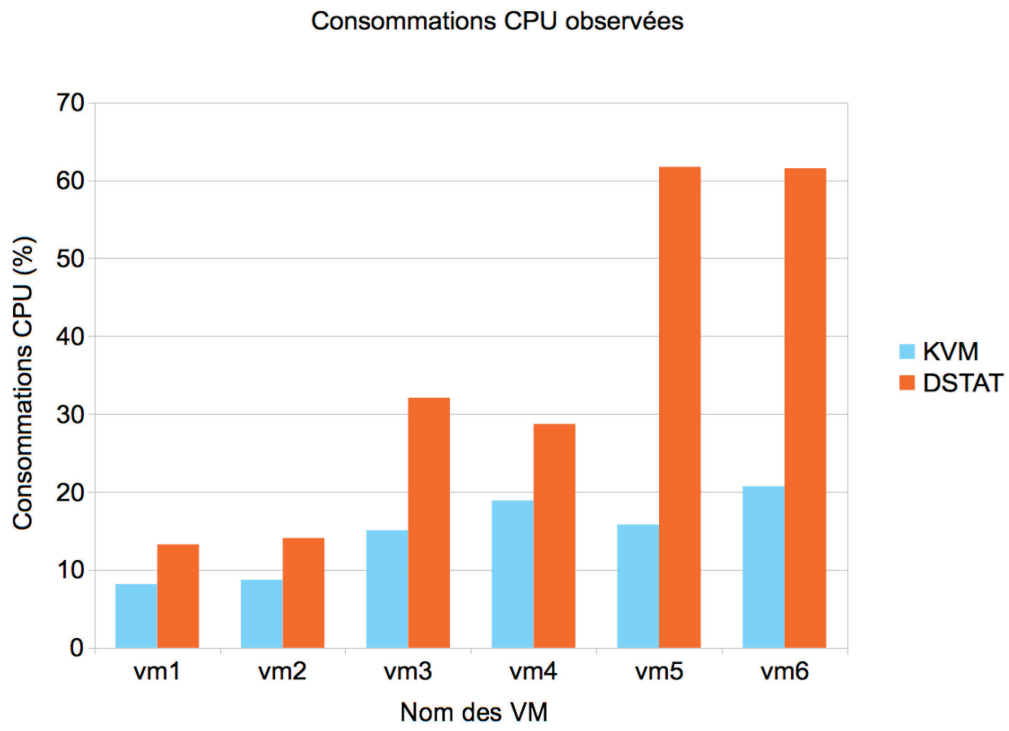

FIGURE 5.5 – Consommations CPU observées à partir de l'hyperviseur KVM et à l'aide de l'outil dstat

- les modules de type « libvirt » permettent la connexion à une infrastructure basée sur des hyperviseurs KVM.

Le module d'exécution est responsable de la parallélisation et de la validation des reconfigurations. Afin de notifier l'administrateur de la fin d'une reconfiguration, le module d'exécution récupère les modifications d'organisation des ressources détectées par le module d'observation. À partir de ces données, le module d'exécution calcule les reconfigurations qui ont été effectuées sur l'infrastructure.

Le gestionnaire de règles est responsable de maintenir à jour la base de règles de placement. Il est capable de lister toutes les règles actives ou inactives et de connaître toutes les règles de placement agissant sur un serveur ou une VM donné.

Le module de placement est capable de calculer les reconfigurations nécessaires afin de mettre en place une politique d'ordonnancement prenant en compte les ressources consommées et les règles de placement. Deux types de module de placement existe :

- le module de type « entropy » utilise la programmation par contraintes pour calculer les reconfigurations à exécuter ;
- le module de type « genetic » utilise la programmation génétique pour calculer les reconfigurations à exécuter.

Les politiques d'ordonnancement pouvant être mises en place par ces modules sont : l'équilibrage de charge, l'économie d'énergie et la correction des problèmes.

Le gestionnaire de temps donne la possibilité à l'administrateur de programmer des reconfigurations ou des règles de placement. Il ajoute une date d'exécution aux reconfigurations et une date d'activation et/ou de désactivation aux règles de placement.

5.3.2 Les modules externes pour une solution distribuée

La programmation distribuée consiste à faire tourner une application modulaire sur plusieurs serveurs. Ce paradigme de programmation est souvent utilisé afin de faciliter le passage à l'échelle ou d'améliorer la tolérance aux pannes d'une application. L'architecture distribuée de l'outil d'administration proposé est mise en place via l'utilisation de modules externes. Ces modules fournissent les mêmes fonctionnalité que les modules internes et communiquent avec le module principal au travers un communication réseau de type TCP/IP. De ce fait, les modules externes peuvent être exécutés sur des serveurs différents. L'utilisation d'une architecture distribuée permet, entre autres : d'augmenter la tolérance aux pannes de l'application, de faciliter le passage à l'échelle, de s'affranchir des limites du langage Java.

Le choix du langage permet, dans certains cas, d'accroître les performances d'un module. Par exemple, certaines librairies comme le VMWare SDK ou libvirt sont plus efficaces dans leur langage natif. Par exemple, la librairie libvirt est écrite en C. L'utilisation de cette librairie à partir du langage Java passe par l'utilisation de la librairie JNA (Java Native Access, en anglais) et entraîne une perte d'efficacité. Le choix du langage de programmation permet aussi de choisir des langages plus adaptés au module à programmer. Par exemple, un interface web se programme plus facilement en PHP qu'en langage C.

Le protocole TCP/IP a été choisi pour la communication entre VMScript et ses modules. Le protocole de communication entre l'outil d'administration et ses modules est simple et l'utilisation de technologie de communication (comme OSGI, RMI, WSDL) ajouterait une complexité en apportant très peu d'avantages. De plus, l'autonomie des modules et le choix de ne pas fixer de langage d'implémentation rend impossible l'utilisation de technologies comme OSGI ou RMI présentes seulement dans le monde Java.

L'interface utilisateur permet d'agir sur le modèle. Quatre types d'opérations sont accessibles : les requêtes, les affectations, les règles et les reconfigurations. Les requêtes sont utilisées pour obtenir des informations sur l'infrastructure comme lister les VM hébergées par un serveur. Les affectations permettent d'enrichir le modèle avec de nouvelles propriétés ou de modifier les propriétés existantes. Les règles définissent le placement des VM sur les serveurs. Les reconfigurations permettent la modification de l'organisation des ressources de l'infrastructure [SA10] comme l'arrêt de VM. Deux interfaces existent : une console texte et une interface web. La console texte propose une interface légère pour l'administration. L'interface web propose d'interagir avec l'outil d'administration à partir d'une interface graphique depuis un navigateur internet. Ce mode permet une visualisation de l'organisation de l'infrastructure et un suivi de l'évolution des ressources consommées dans le temps. L'utilisation d'un module externe pour l'interface graphique permet le choix d'un langage de programmation approprié par rapport à l'interface souhaitée et une administration collaborative entre plusieurs administrateurs. Dans ce cas, chaque administrateur possède une interface lui permettant d'interagir avec l'outil d'administration qui assure notamment à chaque administrateur de visualiser l'évolution de l'organisation des ressources.

L'utilisation de l'outil d'administration et de plusieurs utilisateurs est possible grâce à des communications fréquentes entre l'outil et les différentes interfaces utilisateur proposées. Afin que l'interface soit avertie des modifications intervenant sur le modèle, cette dernière doit s'enregistrer auprès de l'outil d'administration. L'enregistrement est effectué par la requête suivante :

```
REG+num_port
```

Le paramètre « num_port » correspond au port de l'interface sur lequel les informations doivent être envoyées. Une fois enregistré, l'interface reçoit périodiquement les notifications suivantes :

Notifications	Signification
BEG+reconfiguration_notif	Début d'une reconfiguration
END+reconfiguration_notif	Fin d'une reconfiguration
FAI+reconfiguration_notif+message	Échec d'une reconfiguration
ADD+VM+server	Ajout d'une VM à un serveur
REM+VM+server	Suppression d'une VM d'un serveur
LIF+VM_ou_server+nouv_cycle	Changement de cycle de vie d'une VM ou d'un serveur
CHA+element+PRO+nb_prop+prop1+val1+ ...+propN+valN	Changement ou ajout d'une propriété à un élément de l'infrastructure

L'interface communique ensuite avec l'outil d'administration par l'intermédiaire du langage dédié à l'administration. L'envoi d'une requête est effectuée avec la syntaxe suivante :

```
QUE+vmscript_query
```

Par exemple, la récupération de toutes les VM de l'infrastructure est effectuée par la requête suivante :

```
QUE+\vm
```

Le module de reconfiguration externe offre les mêmes fonctionnalités que le module interne. Un pilote informatique (driver, en anglais) est défini comme un programme interagissant avec un périphérique. En ce sens, le module de reconfiguration, qui permet à l'outil d'administration d'interagir avec l'infrastructure, peut être qualifié de pilote. Les pilotes étant les outils contenant le plus de bogues [CYC+01, SABL06], la sortie du module de reconfiguration de l'outil d'administration assure une plus grande stabilité de l'application.

Une phase d'initialisation entre l'outil d'administration et le module de reconfiguration est nécessaire. Lors de cette phase, l'outil envoie la requête suivante :

```
AGENT+vmscript_port
```

Cette requête indique sur quel port le module doit envoyer les notifications d'échec de reconfigurations. À cette requête, le module répond par la liste des serveurs qu'il administre. Grâce à cette liste, l'outil d'administration peut utiliser plusieurs modules de reconfigurations en aiguillant les reconfigurations demandées par l'administrateur vers le bon module de reconfiguration. La gestion de plusieurs modules de reconfigurations est complètement transparente pour l'administrateur. La communication entre l'outil et le module de reconfiguration utilise des requêtes ayant la syntaxe suivante :

Tous les modules de reconfiguration ne prennent pas en compte toutes les requêtes définies ci-dessus. Par exemple, la reconfiguration forçant une certaine consommation de la ressource CPU est seulement utile lors de tests où une activité CPU doit être simulée. Dans le cas de fonctionnalités absentes, la reconfiguration demandée échouera.

Notifications	Signification
STAPM+serveur	Démarrage d'un serveur
STOPM+serveur	Arrêt d'un serveur
SUSPM+serveur	Mise en pause d'un serveur
RESPM+serveur	Sortie de pause d'un serveur
REBPM+serveur	Redémarrage d'un serveur
DELPM+serveur	Arrêt d'administration d'un serveur
CREVM+serveur+vm+template +vcpu_nb+mem	Création d'une VM sur un serveur donné
STAVM+vm+server	Démarrage d'une VM hébergée sur un serveur donné
STOVM+vm+serveur	Arrêt d'une VM hébergée sur un serveur donné
SUSVM+vm+serveur	Mise en pause d'une VM hébergée sur un serveur donné
RESVM+vm+serveur	Sortie de pause d'une VM sur un serveur donné
REBVM+vm+serveur	Redémarrage d'une VM hébergée sur un serveur donné
DELVM+vm+serveur	Arrêt d'administration d'une VM hébergée sur un serveur donné
HARVM+vm+serveur	Force l'arrêt d'une VM hébergée sur un serveur donné
SNAVM+vm+serveur	Sauvegarde d'une VM sur un serveur donné
MIGVM+source+vm+dest	Migration d'une VM hébergée sur un serveur source vers un serveur destination
CPU+vm_ou_serveur+valeur	Force une consommation CPU de « valeur » %
MEM+vm_ou_serveur+valeur	Force une consommation MEM de « valeur » %

Le module d'observation externe proposent les mêmes fonctionnalités que le module d'observation interne. Comme le module de reconfiguration, le module d'observation est un pilote et donc la sortie de ce module de l'outil d'administration assure une plus grande stabilité de l'application. La phase d'initialisation entre l'outil et le module d'observation permet la récupération de l'organisation statique des ressources de l'infrastructure. Cette phase est initiée par l'outil d'administration avec la requête suivante :

```
MON+vmscript_port
```

Deux interprétations sont possibles lors de la récupération de l'organisation statique de l'infrastructure par l'outil d'administration. Dans le premier cas, l'outil possède un modèle construit à partir d'une description textuelle de la grille. L'outil vérifie alors les ressources statiques du modèle comme les capacités CPU des serveurs avec les informations qu'il reçoit. Les serveurs présents dans les informations remontées par le module d'observation mais absents du modèle sont ignorés. Dans le deuxième cas, aucun modèle n'est construit. L'outil construit alors un modèle à partir des informations fournies par le module d'observation. Une fois la phase d'initialisation terminée, le module d'observation envoie périodiquement à l'outil d'administration toutes les modifications dans l'organisation des ressources qu'il détecte. Les modifications sont envoyées à l'aide de requêtes ayant la syntaxe suivante :

À partir de ces quatre notifications, toutes les reconfigurations peuvent être détectées par le module d'observation. Les cas les plus complexes sont la migration et la création de VM.

Dans le cas de la migration, la VM va être supprimée du serveur source pour être ajoutée au serveur de destination. Les deux notifications doivent être envoyées dans le même requête. Si ce n'est pas le cas, la migration sera bien effectuée sur le modèle mais le module d'exécution avertira

Notifications	Signification
ADD+VM+server	Ajout d'une VM à un serveur
REM+VM+server	Suppression d'une VM d'un serveur
LIF+VM_ou_server+nouv_cycle	Changement de cycle de vie d'une VM ou d'un serveur
CHA+element+PRO+nb_prop+prop1+val1+ ...+propN+valN	Changement ou ajout d'une propriété à un élément de l'infrastructure

l'administrateur que la migration est incomplète. À la notification suivante, l'outil d'administration avertira les utilisateurs qu'une modification de l'organisation non attendue s'est produite. La cohérence entre le modèle et l'infrastructure sera donc conservée.

Dans le cas d'une création de VM, la notification d'ajout de VM devra être suivie, dans la même requête, d'une notification de changement de propriétés. Cette deuxième notification devra spécifier les ressources statiques de la VM, c'est-à-dire, sa capacité mémoire ainsi que son nombre de VCPU. Si cette notification n'est pas émise, la VM ne sera pas créée et donc ne pourra être administrée. La cohérence entre le modèle et l'infrastructure ne sera donc pas conservée.

5.4 Tolérance aux pannes

L'utilisation d'une architecture distribuée par l'outil d'administration permet d'augmenter la tolérance aux pannes de l'application. En cas de problème à l'intérieur d'un module externe (bogue provoquant l'arrêt du module, panne sur le serveur), celui-ci peut être redémarré sur le même serveur ou sur un autre serveur. Cependant, le redémarrage d'un module externe n'est pas suffisant pour rétablir la connexion entre l'outil et le module en question. Par exemple, dans le cas d'un module d'observation, la phase d'initialisation permet au module d'enregistrer l'adresse IP et le port utilisé par le serveur TCP afin de lui communiquer ultérieurement les données d'observation. Lors du redémarrage du module, ces informations sont perdues. La phase d'initialisation, initiée par le serveur TCP responsable de la communication avec les modules externes, doit être effectuée de nouveau. Afin de détecter la perte d'une communication entre le serveur TCP et un module externe, le serveur TCP vérifie régulièrement l'état du module en envoyant la requête suivante :

```
PING
```

La liste des modules à contacter est établie par les propriétés « vmscript.agent » et « vmscript.monitor » du fichier de configuration de l'outil d'administration.

Si la connexion au module échoue, le module est supprimé des modules enregistrés par l'outil d'administration. La suppression d'un module peut engendrer des pertes de fonctionnalités. Par exemple, dans le cas d'un module de reconfiguration, les reconfigurations s'exécutant sur les serveurs ou les VM anciennement gérés par ce module ne pourront être exécutées. En cas de problème sur cette partie de l'infrastructure, l'administrateur n'aura aucun moyen d'action.

Si la connexion réussie, le module répond à cette requête avec un numéro d'identifiant unique. De cette manière, le serveur TCP est capable d'identifier si le module est bien celui qu'il a préalablement enregistré. Si ce n'est pas le cas, l'enregistrement du module a lieu et la phase d'initialisation commence.

Le redémarrage de modules externes ayant connu un problème est à la charge de l'administrateur. En effet, l'outil d'administration détecte la perte de connexion qu'il notifie à l'administrateur

mais ne prend pas en charge la réparation du problème. La redéploiement d'un module externe par l'outil est cependant envisageable. Par exemple, l'outil d'administration pourrait tenter de joindre le serveur sur lequel le module s'exécutait avant la panne. Si le serveur est joignable, l'outil peut redémarrer le module en lui donnant la liste des serveurs à gérer. Si le serveur ne répond pas, l'outil peut choisir un serveur dans l'infrastructure sur lequel il déploiera puis démarrera le module perdu.

5.5 Passage à l'échelle

Le passage à l'échelle d'une application se définit par la capacité à gérer un grand nombre de serveurs et, dans notre cas, de VM. Avec l'émergence de l'informatique dans les nuages, la taille des infrastructures virtualisées est en constante augmentation. Par exemple, l'infrastructure de Amazon comprend un demi million de serveurs. Afin d'assurer l'administration d'infrastructures de plusieurs milliers de serveurs et de VM, l'architecture de l'outil d'administration doit être capable de manipuler une grande quantité d'éléments.

5.5.1 Gestion d'un grand nombre de serveurs

La gestion d'un grand nombre de serveurs entraîne la gestion d'un grand nombre de VM. Le taux de consolidation, c'est-à-dire le nombre de VM par serveurs, dans les centres de données est en moyenne de 6.

L'identification des points faibles de l'architecture empêchant le passage à l'échelle de l'outil d'administration a été effectuée à partir d'infrastructures virtualisées contenant plus d'un millier d'éléments. Ces expériences ont permis de montrer qu'un seul module d'observation était insuffisant afin de récupérer l'organisation des ressources en un temps approprié. En effet, la récupération des consommations de centaines de serveurs et de VM nécessite une connexion à chaque serveur et une analyse des ressources. Cette opération est donc coûteuse en temps. Comme l'administrateur se base sur l'observation des ressources afin de calculer les reconfigurations nécessaires à la résolution des problèmes et au maintien d'une politique d'ordonnancement, ces données doivent être récupérées régulièrement. Le module d'observation représente alors un frein au passage à l'échelle de l'outil d'administration. En effet, le temps de récupération des données d'observation doit être suffisamment rapide afin que les mesures soient pertinentes. L'approche distribuée, en multipliant les modules d'observation, garantit une récupération des données d'observation suffisamment rapide.

Dans le cas d'une infrastructure de taille conséquente, plusieurs modules d'observation externes peuvent être utilisés. Dans ce cas, chaque module observe une partie de l'infrastructure. Toutes les modifications observées sont alors envoyées au serveur TCP gérant l'ensemble des modules. Chaque modification est reportée sur le modèle qui permet l'agrégation des données provenant des différents modules d'observation. Dans cette architecture, l'administrateur augmente le nombre de modules d'observation suivant le nombre de serveurs qu'il administre et la période de récupération des données souhaitée.

5.5.2 Gestion de l'hétérogénéité : une solution agnostique

La gestion d'un grand nombre de serveurs augmente les possibilités d'avoir à gérer du matériel hétérogène. Dans le cas de la gestion d'une infrastructure virtualisée, l'hétérogénéité se traduit par la récupération de métriques en fonction du matériel utilisé et par la gestion de différents hyperviseurs.

La récupération de métriques liées au matériel de l'infrastructure est effectuée par le module d'observation. Plusieurs modules d'observation peuvent être utilisés simultanément afin de dédier un module d'observation à la récupération d'une métrique précise. Par exemple, un module peut être dédié à la récupération de la puissance électrique consommée en interrogeant des capteurs reliés aux prises d'alimentation et un autre module à la récupération des ressources utilisées par les VM. De plus, chaque module peut fonctionner avec une période de récupération des données différentes. Par exemple, les ressources consommées par les VM doivent être récupérées régulièrement afin d'assurer une administration correcte. Les consommations électriques, pouvant être jugées secondaires, seront relevées moins fréquemment.

Dans des infrastructures de taille conséquente, il est courant d'avoir à administrer des hyperviseurs de type différent. Par exemple, l'utilisation d'hyperviseurs privés comme VMware ESXi et l'utilisation d'hyperviseurs Open Source comme KVM. Dans ce cas, il est nécessaire d'avoir un module de reconfiguration spécifique à l'hyperviseur. Par exemple, sur un hyperviseur VMWare ESXi, l'utilisation de la librairie libvirt n'offre pas toutes les fonctionnalités. L'ajout de règles de placement n'est pas pris en compte dans libvirt mais est géré à partir du VMware SDK (Software Development Kit, en anglais). L'architecture de l'outil d'administration comprend alors plusieurs modules externes de reconfiguration. Lors de la phase d'initialisation du module, l'outil d'administration lie chaque module de reconfiguration aux serveurs qu'il est capable de reconfigurer. Lors de l'exécution d'une reconfiguration, l'outil récupère le module à joindre à partir du serveur intervenant dans la reconfiguration. La gestion de plusieurs modules de reconfiguration est donc complètement transparente du point de vue de l'administrateur. De plus, l'abstraction offerte par les communications TCP cache à l'outil d'administration le type de l'hyperviseur sous-jacent. La gestion de plusieurs hyperviseurs n'engendre donc pas de configurations supplémentaires de l'outil d'administration.

La figure 5.6 donne un exemple de l'utilisation de l'architecture distribuée de l'outil d'administration. L'infrastructure virtualisée à administrer est composée de deux grappes de serveurs utilisant chacune un hyperviseur. La grappe d'hyperviseurs VMware ESXi est composée de peu d'éléments. Seulement deux modules, un d'observation et un de reconfiguration, sont nécessaires pour son administration. Ces deux modules sont hébergés sur un seul serveur dédié à l'administration de la grappe. Il ne possède donc pas de couche de virtualisation. La grappe d'hyperviseurs KVM nécessite un seul module de reconfiguration mais deux modules d'observation. En effet, le module de reconfiguration est seulement actif lorsqu'une reconfiguration doit être effectuée. Le module d'observation, de son côté, doit communiquer avec les hyperviseurs fréquemment. Ce dernier module consomme donc plus de ressources. Le premier module de reconfiguration surveille une petite partie de la grappe. Sa consommation en ressources est suffisamment basse pour l'exécuter à l'intérieur d'une VM. Le deuxième module de reconfiguration surveillant un plus grand nombre d'hyperviseurs, il est déployé sur un serveur ne proposant pas la virtualisation. Tous ces modules communiquent avec le module principal de l'outil d'administration hébergé sur une VM gérée par un hyperviseur VMware. Ce module étant indispensable, la fonctionnalité de tolérance aux pannes de VMware surveille la VM. Ce mécanisme permet de maintenir un double de la VM sur un autre hyperviseur. En cas de défaillance de la première VM, le double prend sa place et il n'y a pas de perte d'information. Une interface utilisateur est disponible sur un autre serveur permettant à l'administrateur la gestion de l'infrastructure dans son ensemble.

5.5.3 Réactivité et asynchronisme

Le passage à l'échelle d'une application est aussi mesurée en fonction de sa réactivité. La réactivité d'une application se définit par l'intervalle de temps séparant la demande de l'utilisateur

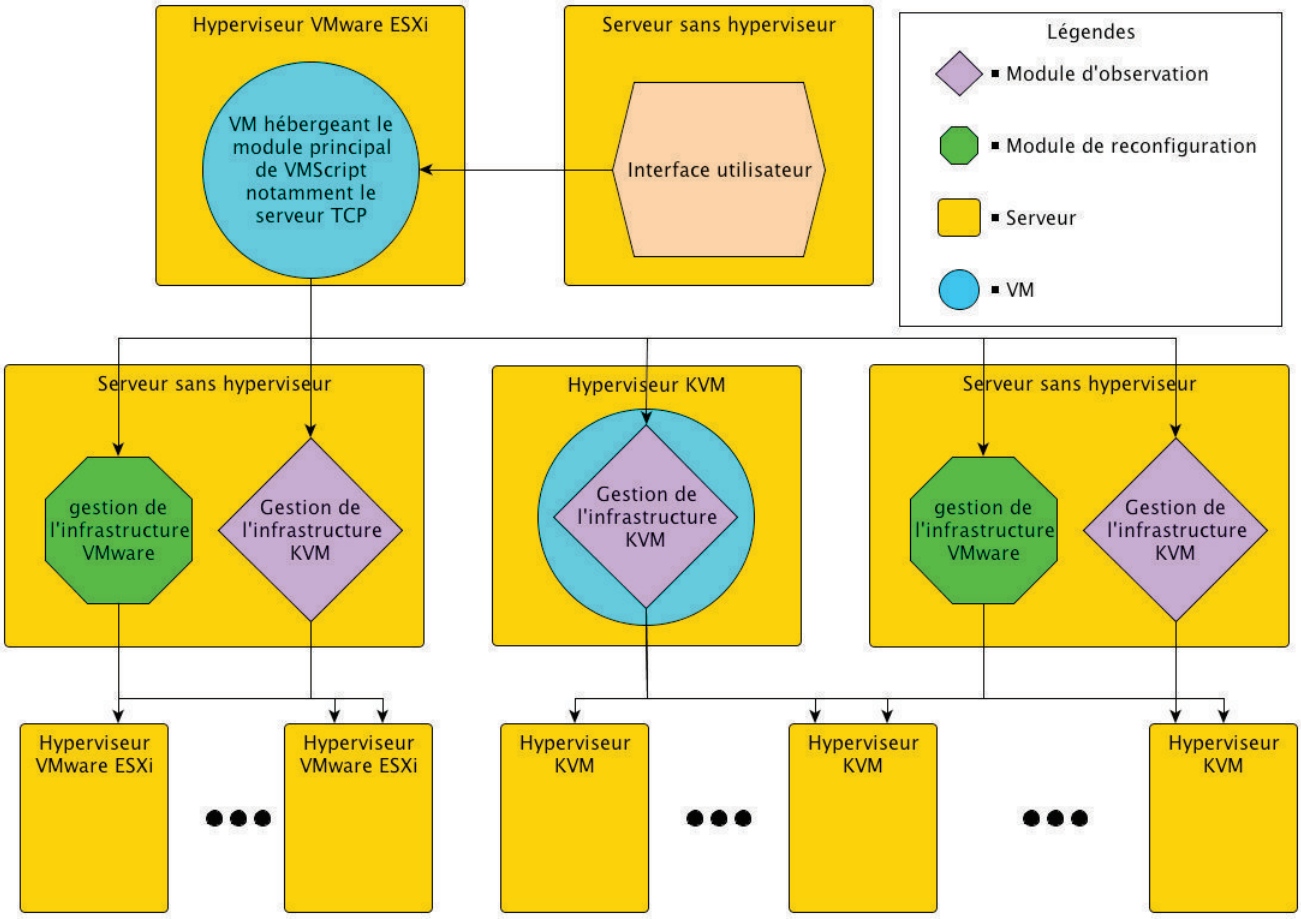

FIGURE 5.6 – Exemple d'architecture distribuée pour l'outil d'administration VMScript

et son exécution. Dans le cas d'un outil d'administration, la réactivité dépend du temps écoulé entre la demande de reconfiguration de l'administrateur et l'exécution de cette reconfiguration sur l'infrastructure.

Dans une infrastructure de taille conséquente, l'administrateur ne peut attendre la fin d'une reconfiguration avant de pouvoir continuer à administrer l'infrastructure. Par exemple, lorsque l'administrateur arrête une VM, l'application ne doit pas attendre l'arrêt de la VM pour rendre la main à l'administrateur. Toutes reconfigurations possèdent un temps d'exécution qui est variable en fonction de la VM. Par exemple, le temps d'arrêt d'une VM dépend du système d'exploitation qu'elle utilise mais aussi des applications qu'elle exécute et des ressources disponibles sur l'hyperviseur l'hébergeant. Afin d'éviter de bloquer l'administrateur lors de reconfigurations, les reconfigurations de l'outil d'administration proposées sont toutes asynchrones. L'administrateur peut donc constamment administrer l'infrastructure. Cependant, l'administrateur a besoin de savoir quand une reconfiguration est finie et si il n'y a pas eu d'erreur lors de son exécution. Pour cela, toutes les modifications de l'infrastructure relevées par le module d'observation sont envoyées au module d'exécution de l'outil d'administration. Ce dernier peut donc avertir l'administrateur lorsqu'une reconfiguration est terminée. L'administrateur peut alors vérifier que la reconfiguration a bien résolu son problème. Si une erreur survient lors d'une reconfiguration, le module de reconfiguration envoie un message afin de prévenir l'administrateur. L'administrateur peut donc envoyer une reconfiguration au module d'exécution même si une reconfiguration est en cours.

La syntaxe du langage et la gestion asynchrone des reconfigurations permettent à l'administrateur d'envoyer simultanément plusieurs reconfigurations au module d'exécution. Le module d'exécution va alors gérer les reconfigurations d'après un algorithme de « Premier arrivé, premier servi » (« First In, First Out », en anglais). Afin d'améliorer la réactivité de l'outil, le module d'exécution va essayer d'exécuter les reconfigurations le plus tôt possible en les parallélisant. La parallélisation des reconfigurations permet d'exécuter un certain nombre de reconfigurations en même temps. Cependant, la parallélisation nécessite quelques précautions. Par exemple, si l'administrateur demande la migration de deux VM hébergées par un serveur et l'arrêt de ce serveur, le module d'exécution doit attendre les fins des migrations avant d'arrêter le serveur. Afin d'éviter ce genre de problèmes, une liste d'acteurs est définie pour chaque reconfiguration. Les acteurs d'une reconfiguration sont tous les serveurs et VM intervenant dans une reconfiguration. Par exemple, la migration d'une VM définit trois acteurs : le serveur hébergeant la VM, la VM et le serveur recevant la VM. L'arrêt d'un serveur définit un seul acteur, le serveur devant être arrêté. Si deux reconfigurations possèdent au moins un acteur en commun, les reconfigurations ne peuvent être parallélisées. Cette méthode permet d'éviter le problème décrit précédemment. De plus, une reconfiguration consomme des ressources. Par exemple, un migration consomme de la ressource CPU et de la ressource réseau. Si plusieurs migrations sont exécutées en parallèle, les migrations risquent d'être plus longues voire d'échouer si la ressource réseau devient très insuffisante.

5.6 Conclusion

L'administrateur doit être capable de détecter un serveur surchargé ou une règle de placement non respectée rapidement afin de maintenir l'organisation des ressources dans un état cohérent. Dans une infrastructure de taille conséquente, des outils sont nécessaires pour mener à bien cette tâche. L'outil d'administration proposé intègre des mécanismes de gestion de la cohérence lors d'insertion de règles de placement et d'exécution de reconfigurations. De plus, à l'aide du module de surveillance, l'organisation des ressources est vérifiée périodiquement afin de détecter

les serveurs surchargés et les règles de placement non respectées.

La gestion de l'environnement d'exécution des langages dédiés est assurée par un outil modulaire. Cette modularité permet d'adapter l'outil à une infrastructure hétérogène en sélectionnant les modules nécessaires à la gestion de plusieurs hyperviseurs. De plus, la modularité facilite la distribution des fonctionnalités afin d'assurer le passage à l'échelle de l'application.

Chapitre 6

Expérimentation

L'expérimentation de l'outil d'administration proposé est effectué en deux étapes : l'évaluation du langage dédié à l'administration et l'évaluation de l'outil d'administration. La première étape compare le langage dédié à l'administration à deux langages : un langage généraliste de script et un langage dédié à la gestion de l'outil d'administration de VMware. Le langage généraliste, Bash, est utilisé pour comparer l'exécution de reconfigurations sur des hyperviseurs. Le langage dédié, PowerCLI, permet d'interagir avec l'outil d'administration VMware vSphere. Ce langage sera utilisé pour comparer la gestion des règles de placement dans vSphere et dans l'outil d'administration proposé dans ce document. Dans les deux cas, l'outil d'administration offre un langage plus concis et des mécanismes garantissant un maintien de la cohérence. Pour les reconfigurations, le maintien de la cohérence est effectué à l'aide des préconditions de chaque reconfiguration. Pour les règles de placement, la cohérence est maintenue par le gestionnaire de règle garantissant une base de règles ne contenant pas de contradiction. L'évaluation de l'outil d'administration débute par l'évaluation du module de placement utilisé pour résoudre les incohérences détectées sur l'infrastructure. Les différents paramètres de l'algorithme génétique d'ordonnancement sont alors détaillés et configurés pour assurer une recherche optimale du plan de reconfigurations. Dans un second temps, l'outil d'administration est déployé sur une infrastructure de taille conséquente. L'utilisation de la plateforme grid5000 est alors utilisée pour construire une infrastructure virtualisée et étendue sur plusieurs villes.

Sommaire

6.1 Évaluation du langage d'administration

L'évaluation du langage dédié à l'administration porte sur la comparaison de ce dernier avec deux autres langages. La première comparaison étudie l'exécution d'une reconfiguration avec le langage d'administration proposé et avec le langage généraliste Bash. La deuxième comparaison se focalise sur la gestion des règles de placement avec le langage VMware PowerCLI et avec le langage proposé dans ce document.

6.1.1 Comparaison avec un langage généraliste

Bash est l'interpréteur de commandes par défaut des systèmes GNU/Linux. Cet outil est très populaire pour l'administration de serveurs. Dans l'exemple suivant, l'administrateur arrête les hyperviseurs de type Xen n'hébergeant pas de VM et consommant moins de 10 % de ressource CPU. L'arrêt de serveurs est nécessaire pour effectuer des tâches de maintenance comme l'ajout de mémoire ou le changement d'un disque dur.

Le script Bash ci-dessous récupère la version du SE avec la commande « uname -r » et la consommation CPU de l'hyperviseur. Ensuite, un test filtre les hyperviseurs pour ne conserver que les hyperviseurs Xen. Pour finir, une conditionnelle permet d'arrêter uniquement les hyperviseurs n'hébergeant pas de VM.

```
for server in $* ; do
  kernel=$(ssh $server uname -r)
  cpu=$(ssh $server dstat -c 1 1|tail -n 1|awk '{ print $1+$2 }')
  if [ $cpu = "2000" -a $kernel = "2.6.26-1-xen-amd64" ]; then
    if [ -z $(ssh $server xm li | sed '1d' | sed '1d')" ]; then
      ssh $server halt
    fi
  fi
done
```

La requête du langage dédié à l'administration ci-dessous sélectionne tous les serveurs et applique deux filtres à la sélection. Le premier utilise la propriété « se » pour supprimer les serveurs n'utilisant pas l'hyperviseur Xen et le deuxième filtre utilise la propriété « cpu_cons » pour ne conserver que les hyperviseurs ayant une basse consommation CPU et donc peu utilisés. Pour finir, la reconfiguration « stop » arrête les serveurs. La précondition de la reconfiguration permet d'empêcher l'arrêt d'un hyperviseur hébergeant des VM.

```
/server{cpu_cons < 10 & se == xen}:stop
```

Le langage proposé offre à l'administrateur :

- une requête plus concise. La même reconfiguration est décrite en une ligne au lieu de 9 ;
- un gain d'efficacité. Une seule connexion réseau est utile avec le langage dédié à l'administration contre trois connexions ssh avec le script Bash. En effet, les informations nécessaires sont récupérées dans le modèle maintenu à jour par l'outil d'administration ;
- une requête hyperviseur agnostique. Le script Bash utilise l'API (Application Programming Interface, en anglais) de l'hyperviseur Xen pour vérifier la présence éventuelle de VM. Cette partie doit être modifiée si l'hyperviseur est différent. La requête du langage dédié est identique quelque soit l'hyperviseur utilisé.

6.1.2 Comparaison avec un vSphere API

La solution de virtualisation VMware vSphere propose des règles d'affinité similaires aux règles « group » de l'outil d'administration proposé et des règles d'anti-affinité similaires aux règles « spread ». L'interpréteur de ligne de commande PowerCLI basé sur du PowerShell offre un langage de script de gestion de l'outil vSphere. L'exemple suivant compare la gestion d'une règle de placement avec PowerCLI et avec le langage proposé dans ce document. La règle à mettre en place doit empêcher les VM « proxy1 », « proxy2 », « proxy3 » de s'exécuter sur le même serveur. Deux autres VM « proxy4 » et « proxy5 » existent sur l'infrastructure et compliquent l'utilisation des expressions régulières.

Dans l'outil vSphere, les serveurs sont regroupés en grappe et une règle est associée à une grappe. La requête PowerCLI ci-dessous crée la règle souhaitée :

```
New-DrsRule -Name Proxy -Cluster cl1 KeepTogether:$false -VM Proxy1, Proxy2, Proxy3
```

Avec PowerCLI, la grappe contenant la règle est spécifiée par le paramètre « -Cluster ». Le paramètre « -Name » détermine le nom de la règle et le paramètre « -VM » permet de sélectionner les VM à partir de leur nom. L'option « KeepTogether » est mise à faux afin de séparer les VM sur plusieurs serveurs.

Dans le langage dédié, les règles ne sont pas associées à une grappe mais à toute l'infrastructure. Il n'est donc pas nécessaire de spécifier la grappe contenant les serveurs. L'opérateur « spread » remplace le paramètre « KeepTogether » et le mécanisme de sélection du langage d'administration permet une expression concise du nom des VM :

```
Proxy: proxy[1-3] spread
```

La commande « New-DrsRule » permet à l'administrateur d'activer ou de désactiver une règle lors de sa création ce que ne permet pas l'outil d'administration proposé. En effet, toutes les règles non programmées sont activées dès lors définition. L'administrateur peut cependant modifier l'état de la règle après sa définition :

```
Proxy: proxy[1-3] spread
Proxy:disable
```

Après la création d'une règle, celle-ci peut être modifiée afin de changer son état ou les VM et les serveurs qu'elle utilise. Avec PowerCLI, l'ajout de la VM « vm4 » passe par une redéfinition complète de l'ensemble de VM :

```
Set-DrsRule -Rule Proxy -VM Proxy1,Proxy2,Proxy3,Proxy4
```

Avec le langage dédié, la VM est ajoutée à l'ensemble déjà existant :

```
proxy:vm + proxy4
```

Si une règle ajoutée est contradictoire avec une règle existante, vSphere la désactive sans en informer l'administrateur. Dans l'outil d'administration proposé, l'ajout d'une règle entraînant une contradiction provoque l'envoi d'un message à l'administrateur et annule l'ajout de la règle.

Les règles d'affinité entre VM et serveurs (correspondant aux règles « on » et « noton ») sont aussi disponibles dans l'outil vSphere. Cependant, leur manipulation avec PowerCLI n'est pas triviale [vL12]. De plus, contrairement à l'outil proposé, il n'y a pas de détection de conflit pour

ce type de règles dans vSphere. L'administrateur peut donc ajouter une règle et son contraire dans la base de règles.

L'exécution des reconfigurations dans vSphere est indépendante des règles d'affinité. Une reconfiguration ne respectant pas une ou plusieurs règles peut donc être exécutée ce qui n'est pas possible dans l'outil d'administration proposé.

Pour conclure, l'outil d'administration proposé et l'outil PowerCLI associé à vSphere proposent une approche similaire permettant de gérer des règles de placement à partir des noms des VM et des serveurs. Cependant, la gestion des règles fournit par l'outil proposé dans ce document est plus concise et plus sûre grâce à la détection de conflits pour toutes les règles de placement. De plus, l'outil proposé utilise les règles de placement pour empêcher toutes reconfigurations incohérentes.

6.2 Évaluation du système du maintien de la cohérence

Cette section décrit une évaluation du système de maintien de la cohérence. En pratique, le système de détection d'erreurs ne propose pas de limite réelle et le mécanisme d'aide réalisé par Entropy a été effectué et tester dans la thèse de Fabien Hermenier intitulée : « Gestion dynamique des tâches dans les grappes, une approche à base de machines virtuelles ». Cette section est donc dédiée à l'étalonnage de l'algorithme génétique présenté dans le chapitre précédent.

6.2.1 Étalonnage de l'algorithme génétique

L'étalonnage d'un algorithme génétique consiste à mesurer l'impact de chaque variable sur la solution obtenue. Chaque variable est fixée à une valeur basse de sa plage de valeurs. Les plages de valeurs de chaque variable sont ensuite parcourues et la qualité des solutions apportées est mesurée. Ces mesures prennent en compte le nombre de problèmes non résolus dans la meilleure solution obtenue et le nombre de reconfigurations nécessaires pour obtenir l'organisation des ressources souhaitée. Les valeurs obtenues correspondent à une moyenne de cinq cent mesures (les valeurs sont disponibles à l'annexe 8.1). La solution optimale contient 0 problème et un coût de reconfiguration de 3. Les valeurs initiales utilisées pour la configuration de l'algorithme sont les suivantes :

Variables	Valeurs
Nombre de générations (x nombre de VM)	1
Taille de la population (x nombre de VM)	1
Nombre de reproduction (% de la population)	50
Nombre de mutation (% de la population)	20
Nombre de gènes modifiés par mutation (% du nombre de VM)	10

Le nombre de générations (voir figure 6.1) influence peu les solutions obtenues. Un nombre de générations trop restreint dégrade néanmoins la qualité du plan de reconfigurations obtenu.

Une taille de population moyenne diminue le nombre de problèmes mais augmente le nombre de reconfigurations (voir figure 6.1). Les résultats sont identiques pour les grandes et les moyennes

valeurs. Cependant, plus la taille de la population est importante, plus le calcul de la solution est long car il y a plus de reproductions et de mutations à effectuer. Il faudra donc veiller à ne pas utiliser une taille trop grande.

Le nombre de reproductions n'influe pas sur la résolution des problèmes (voir figure 6.3). Cependant, un nombre trop petit de reproductions entraîne une hausse du nombre de reconfigurations.

Le nombre de mutations (voir figure 6.2) ainsi que le nombre de gènes modifiés par mutation (voir figure 6.2) n'ont pas d'influence sur le nombre de problèmes détectés. Ces critères sont néanmoins utiles afin de diminuer le coût du plan de reconfigurations.

Les reproductions et surtout les mutations sont les paramètres les plus pertinents pour réduire le coût du plan de reconfigurations. La résolution du nombre de problèmes est, quant à elle, influencée par la diversité de la population représentée par sa taille.

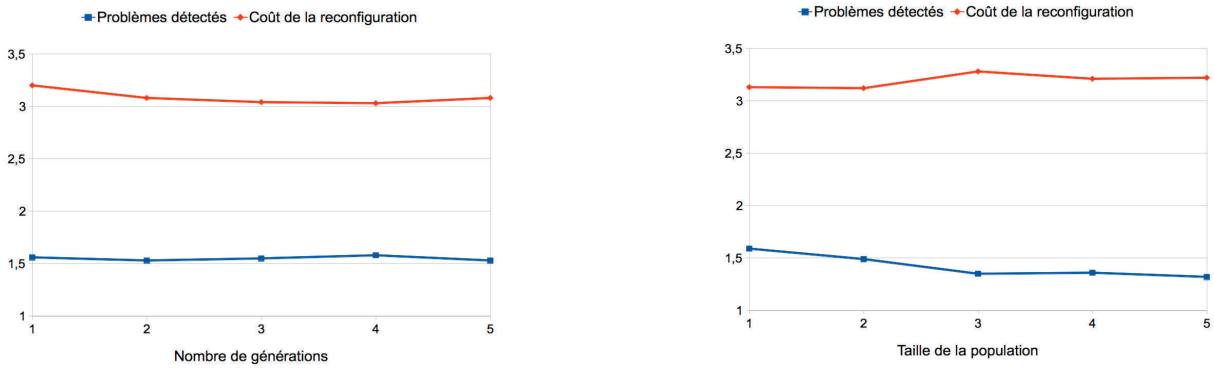

FIGURE 6.1 – Influence du nombre de générations et de la taille de la population

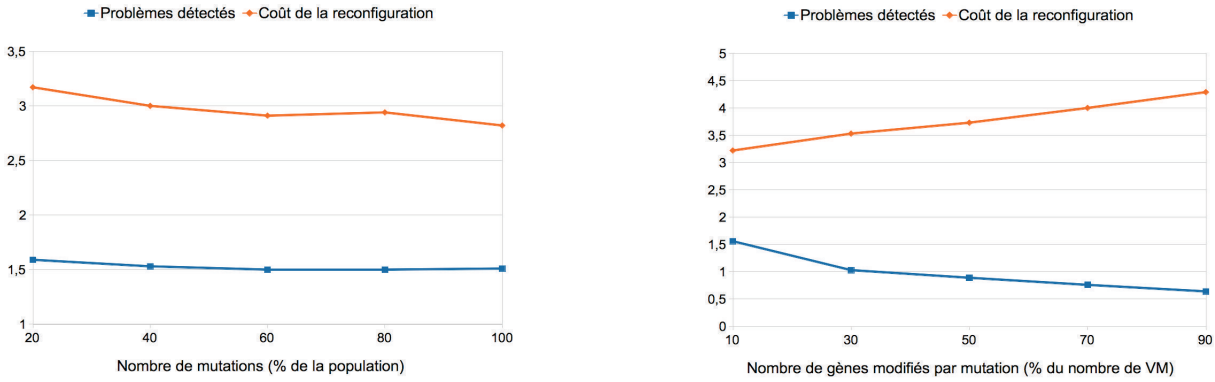

FIGURE 6.2 – Influence du nombre de mutations et du nombre de gènes modifiés par mutation

Une fois l'impact des différents paramètres mesuré, un étalonnage plus précis est réalisé pour trouver les valeurs optimales de chaque variable. L'optimisation commence par fixer une valeur proche de l'optimale pour chaque variable et à effectuer des mesures en faisant varier les variables une à une. Les solutions obtenues étant de meilleures qualités, les moyennes observées pour l'évaluation des solutions sont proches des valeurs optimales. L'algorithme est évalué suivant le pourcentage de solution optimale qu'il retourne. Les solutions sont évaluées à partir de deux critères :

– le pourcentage de solutions ayant aucun problème ;

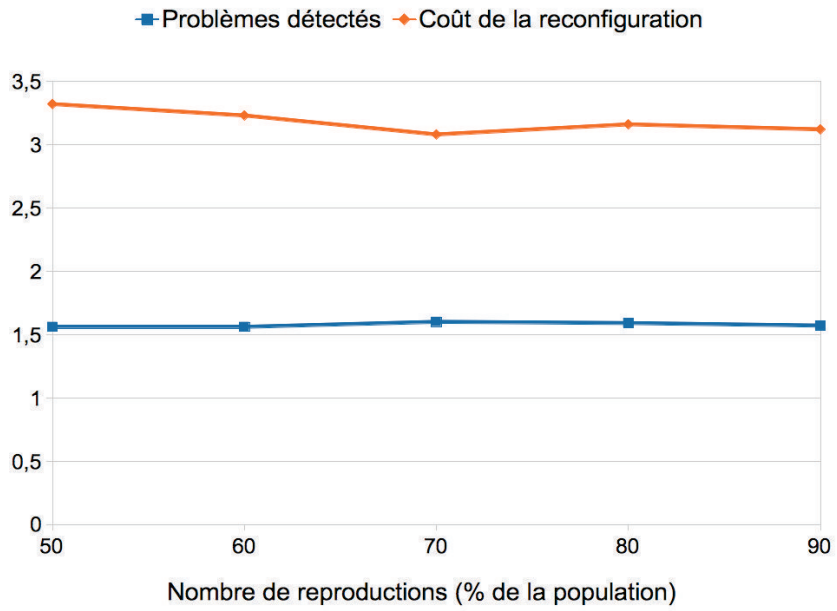

FIGURE 6.3 – Influence du nombre de reproductions

– le pourcentage de solutions ayant un coût de reconfiguration égal à 3.

Les valeurs retenues pour affiner la configuration l'algorithme génétique d'ordonnancement sont les suivantes :

Variables	Valeurs
Nombre de générations (x nombre de VM)	2
Taille de la population (x nombre de VM)	3
Nombre de reproduction (% de la population)	60
Nombre de mutation (% de la population)	60
Nombre de gènes modifiés par mutation (% du nombre de VM)	60

Le nombre de générations influence fortement la qualité du plan de reconfigurations obtenu. Si ce nombre est trop bas, les solutions sont de piètre qualité (voir figure 6.4). Ce paramètre étant celui influençant le plus le temps de calcul, la valeur retenue est 2. La taille de la population doit être assez importante afin de réduire considérablement le coût du plan de reconfigurations (voir figure 6.4). La valeur choisie pour ce paramètre est de 4. Le nombre de reproductions influence légèrement le qualité des plans de reconfigurations calculés (voir figure 6.6). La valeur retenue est de 50 %. Le nombre de mutations permet de réduire le nombre de reconfigurations (voir figure 6.5) sans ajouter de problèmes. Ce paramètre est réglé à 100 %. Le nombre de gènes modifiés par mutation permet de réduire de façon importante les problèmes détectés. Cependant, la valeur de ce paramètre ne peut être trop élevée car elle augmente le coût du plan de reconfigurations (voir figure 6.5). La valeur choisie est de 50 %.

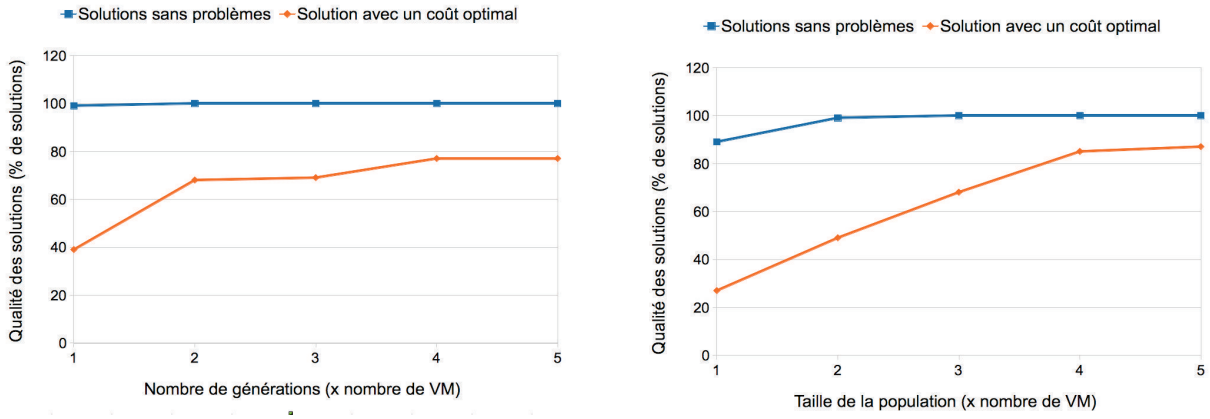

FIGURE 6.4 – Affinage du nombre de générations et de la taille de la population

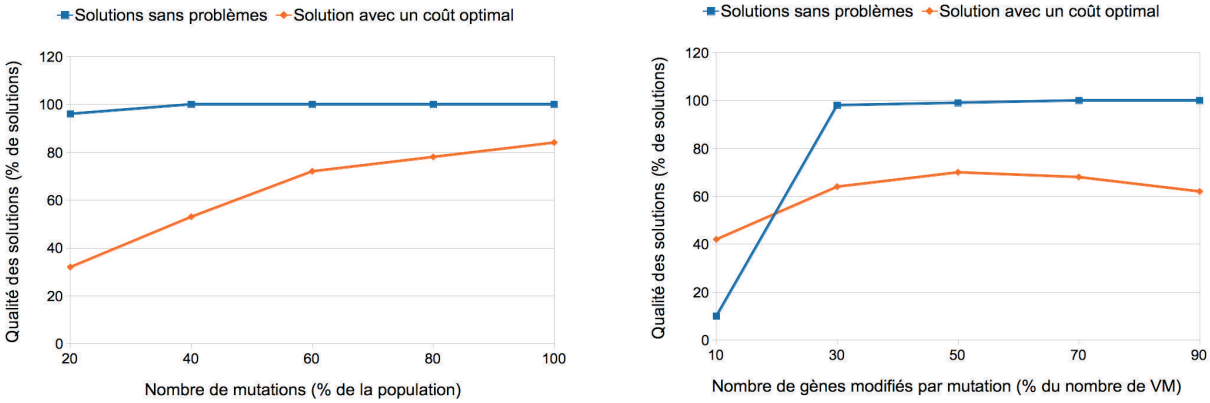

FIGURE 6.5 – Affinage du nombre de mutations et du nombre de gènes modifiés par mutation

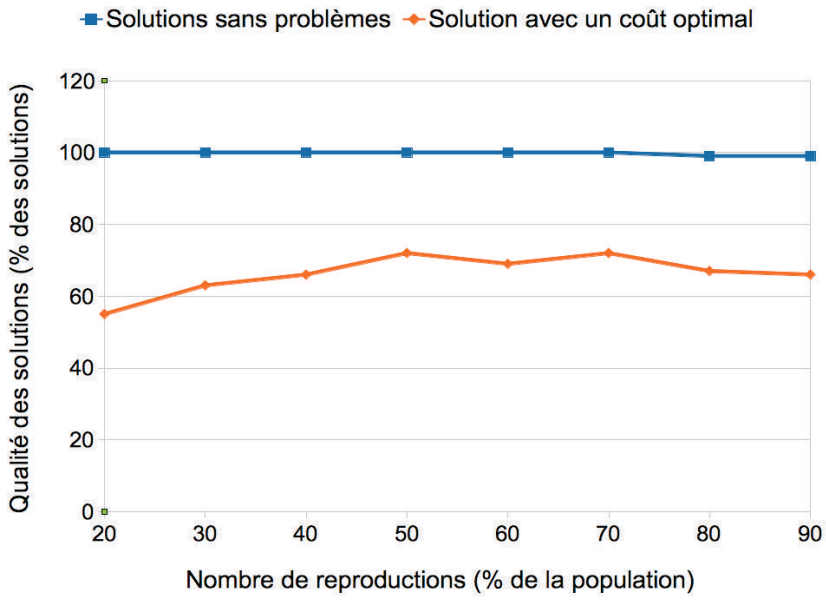

FIGURE 6.6 – Affinage du nombre de reproductions

Pour résumer, les valeurs retenues pour la configuration de l'algorithme génétique sont les suivantes :

Variables	Valeurs
Nombre de générations (x nombre de VM)	2
Taille de la population (x nombre de VM)	4
Nombre de reproduction (% de la population)	50
Nombre de mutation (% de la population)	100
Nombre de gènes modifiés par mutation (% du nombre de VM)	50

6.2.2 Optimisation de l'algorithme

Les algorithmes génétiques nécessitent une population diversifiée afin de parcourir l'ensemble des solutions. Afin d'augmenter la diversité dans l'algorithme présent dans l'outil d'administration proposé, la suppression d'individus identiques est réalisée au moment du classement des individus. Cette optimisation permet une convergence plus rapide vers les solutions optimales.

Solutions sans problème (%)	Solutions avec un coût optimal (%)
100	90
100	97

TABLE 6.1 – Influence de l'optimisation de l'algorithme génétique

6.3 Simulation d'une infrastructure virtualisée de grande taille

Afin de tester l'outil d'administration proposé et ses différents modules, un mode simulation est disponible dans lequel aucune infrastructure n'est utilisée. Le modèle de l'outil d'administration est créé à partir des fichiers texte de description puis toutes les reconfigurations sont effectuées sur le modèle. À l'aide du langage dédié à l'administration, l'administrateur peut simuler une modification des ressources consommées. Par exemple, l'expression « serveur1 :cpu_cons = 40 » simule une consommation CPU de 40 % sur le serveur « serveur1 ». Ce mode est aussi utilisé pour tester facilement les algorithmes d'ordonnancement et le module de surveillance.

Afin de simuler une infrastructure réelle, un module de simulation permet de générer automatiquement une consommation de ressources. Ce module est connecté à l'outil d'administration à la place du module d'observation composé de sondes. À partir de fichiers texte, le module va modifier les consommations CPU et mémoire des hyperviseurs et des VM à des temps donnés. Les modifications des ressources peuvent aussi être faites à partir d'expressions mathématiques utilisant des ressources. Par exemple, la consommation électrique peut être définie en fonction de la consommation CPU du serveur.

6.4 Déploiement de l'outil d'administration sur une infrastructure virtualisée

En vue d'effectuer des tests ou des démonstrations, l'outil d'administration peut être utilisé en mode simulation, c'est-à-dire sans être relié à une infrastructure réelle. Dans ce cas, toutes les reconfigurations sont effectuées sur le modèle. Néanmoins, le mode simulation ne permet pas de tester la robustesse ou le passage à l'échelle de l'outil. L'outil proposé dans ce document se démarque des autres solutions, notamment de celles basées sur une interface graphique, par sa capacité d'administration d'infrastructures virtualisées de taille conséquente à l'aide de langages dédiés. Un test sur une infrastructure réelle est donc nécessaire. Une infrastructure composée d'une centaine d'hyperviseurs répartis dans plusieurs villes n'est pas disponibles directement. L'utilisation de la plateforme grid5000 [CDCG+05] a permis de réaliser un test sur une infrastructure de taille importante. Cette plateforme est une infrastructure géographiquement distribuée sur 10 sites en France. Son but est d'offrir une plateforme expérimentale pour l'étude de systèmes distribués à grande échelle. Nativement, la plateforme ne permet pas l'utilisation de la virtualisation. L'outil VGrid a été développé afin de déployer une infrastructure virtualisée décrite à l'aide du langage dédié à la description d'infrastructure virtualisée sur cette plateforme.

6.4.1 L'architecture de l'infrastructure déployée par VGrid

VGrid tente de fournir une première brique vers un outil de déploiement d'infrastructures virtualisées à la plateforme grid5000. Dans cet objectif, l'outil fournit une infrastructure configurable tentant de répondre aux besoins du plus grand nombre d'utilisateurs.

L'utilisation du langage de description permet de décrire précisément l'infrastructure à l'aide de fichiers texte. Afin de coller à l'infrastructure de grid5000, la vue physique de l'infrastructure est décrite avec la hiérarchie suivante : site, grappe, serveur. Plusieurs propriétés indispensables à la configuration de l'infrastructure sont précisés dans la description. Pour les serveurs, la propriété « env » sélectionne l'environnement de référence mise à disposition par la plateforme grid5000. Cet environnement choisit et configure le système d'exploitation à installer sur le serveur. La propriété « role » permet l'installation et la configuration d'applications liées au rôle du serveur dans l'infrastructure (par ex., l'installation et la configuration de l'hyperviseur). Pour les VM, la propriété « env » correspond à l'image disque de référence à copier pour créer la VM. Les propriétés « mem » et « cpu_nb » désignant respectivement la capacité mémoire et le nombre de vcpu de la VM correspondent aux capacités allouées à la VM lors de son instanciation.

Une infrastructure virtualisée nécessite plusieurs types de composants logiciels :

- des hyperviseurs pour pouvoir créer des VM ;
- des serveurs de stockage pour stocker les images disque des VM ;
- des serveurs de services pour exécuter les outils d'administration de VM (par ex., un outil d'administration).

Dans l'infrastructure déployée (voir figure 6.7), chaque composant est installé sur un serveur différent. L'hyperviseur choisit est Kvm pour sa simplicité d'installation et de configuration sur les systèmes Linux/Debian. La librairie libvirt est aussi installée afin de simplifier la gestion des VM. Chaque VM aura sa propre image disque. Les serveurs de stockage utilisent le protocole NFS (Network File System, en anglais) afin de partager les images disque de VM entre les serveurs d'une même grappe. Le partage des images de VM est nécessaire afin d'effectuer des migrations sans transférer l'image disque de la VM par le réseau. Une solution alternative serait de copier toutes les images de VM sur tous les nœuds de la grappe. Cette solution est très longue car elle implique de nombreuses copies de plusieurs gigabits. La gestion des VM sera faite à l'aide

de l'outil d'administration proposé dans ce document configuré avec des modules externes de reconfigurations et d'observation.

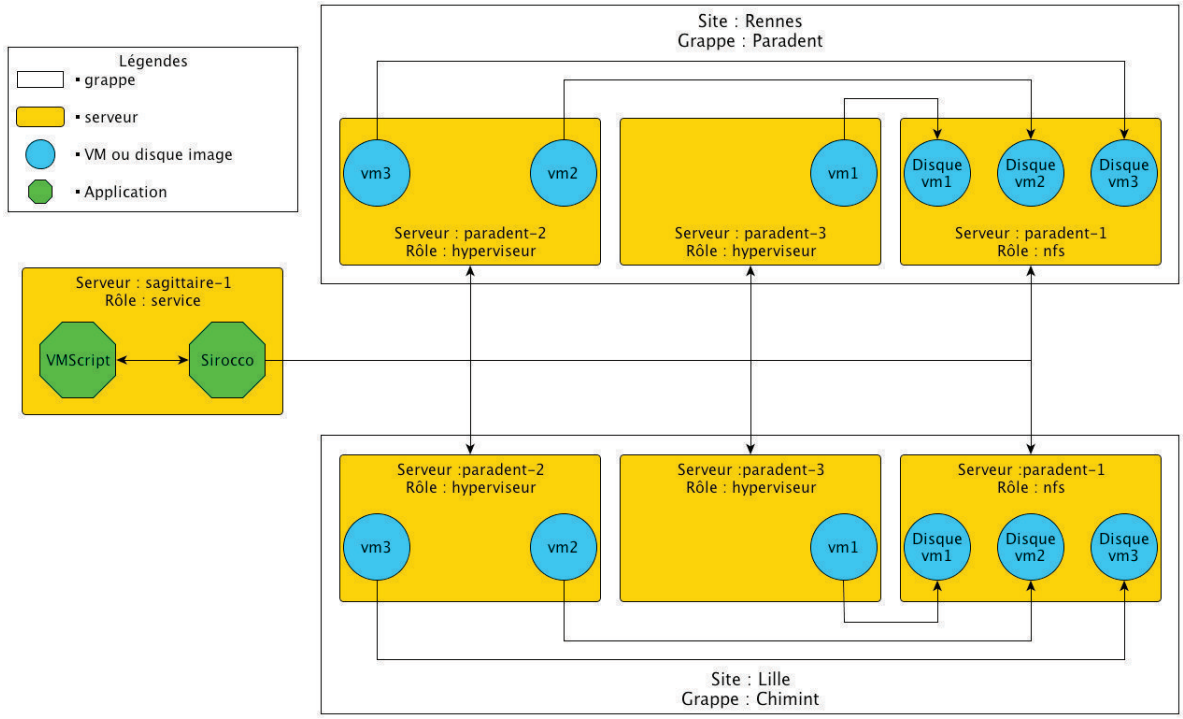

FIGURE 6.7 – Description de l'infrastructure déployée sur Grid5000

La plus grande infrastructure déployée sur Grid5000 comprend plus de deux cents serveurs répartis sur sept sites et hébergeant un millier de VM. Le modèle construit par l'outil d'administration, calqué sur l'infrastructure réelle et son organisation en sites et en grappes, permet de parcourir l'organisation des ressources. Les modules d'exécution et d'observation sont connectés à l'outil d'administration afin d'exécuter les reconfigurations de l'infrastructure et de mettre à jour les ressources consommées par les VM. Le gestionnaire de règles et le module de surveillance garantissent la cohérence de l'infrastructure.

6.4.2 Problèmes observés

Le premier problème observé lors du déploiement de l'infrastructure virtualisée est le temps nécessaire à l'installation et à la configuration des serveurs mais surtout à la création des multiples images disque des VM. Ces images disque ayant une taille de 512 Mo à plusieurs gigaoctets sont longues à copier ce qui entraîne une durée d'environ deux heures pour le déploiement d'un millier de VM. Afin de réduire le temps de déploiement, des images disque creuses (sparse, en anglais) ayant une taille équivalente aux données réellement utilisées peuvent être utilisées. Ainsi, pour une distribution minimale de Linux, l'image disque a une taille de moins de 100 Mo.

Le second problème observé est le manque de réactivité du module d'observation sur une infrastructure d'une taille importante. En effet, l'observation d'un grand nombre de serveurs et de VM ralentis considérablement la récupération des ressources consommées. Il devient alors impossible de connaître les ressources consommées et donc de détecter les problèmes de surcharge sur l'infrastructure. L'utilisation de 3 modules d'observation a donc été nécessaire afin de récupérer les données d'observation en moins d'une minute.

Troisième partie

Conclusion

Chapitre 7

Conclusion et travaux futurs

Afin de répondre à la demande grandissante des services informatiques, les centres de données se développent en termes de taille et de complexité. Avec l'utilisation de la virtualisation, ces infrastructures accueillent plus d'applications et deviennent plus flexibles permettant ainsi à l'administrateur de réorganiser les ressources sans altérer le fonctionnement des applications.

L'outil d'administration présenté dans ce document propose un langage dédié à l'administration d'infrastructures virtualisées. Afin de faciliter l'administration d'infrastructures de taille conséquente, cet outil construit sa représentation de l'infrastructure, le modèle. Ce dernier, décrit à l'aide d'un langage dédié, regroupe l'organisation de l'infrastructure et les propriétés liées à chacun de ses éléments. À l'aide de ces propriétés, l'administrateur sélectionne et manipule des ensembles d'éléments. Des règles de placement définissant des besoins précis sont utilisées pour décrire des relations de placement entre les serveurs et les VM. Par exemple, la mise en place de mécanismes de tolérance aux pannes. Un module de surveillance analyse périodiquement l'infrastructure afin d'alerter l'administrateur en cas de serveurs surchargés ou de règles de placement non respectées. La résolution des problèmes détectés est alors effectuée par des reconfigurations cohérentes effectuées par l'administrateur ou par l'appel au module de placement qui calcule un plan de reconfigurations. Ce dernier module est aussi utilisé par l'administrateur pour mettre en place des politiques d'ordonnancement. Grâce à l'utilisation du langage, l'administrateur peut appliquer différentes politiques comme la consolidation ou l'équilibrage de charge sur des parties différentes de l'infrastructure.

Les ressources utilisées par les VM sont nombreuses : réseau, mémoire, cpu... Dans l'outil d'administration proposé, seules les ressources CPU et mémoire sont prises en compte dans les phases de surveillance ou de reconfigurations. La prise en charge de ressources supplémentaires sont nécessaires afin de fournir un outil d'administration complet. Par exemple, la prise en charge de la ressource disque est essentielle pour connaître les images disque des VM disponibles sur un hyperviseur et permettre ou refuser une migration à chaud.

L'outil d'administration présenté dans ce document n'est pas complètement tolérant aux pannes. En effet, l'architecture distribuée concerne seulement les modules externes. La brique principale responsable du traitement des requêtes de l'administrateur et de la communication entre les modules reste centralisée. En cas de panne de l'outil d'administration ou du serveur l'hébergeant, le modèle ainsi que toutes les règles de placement mises en place sont perdues. Pour limiter le problème, les règles de placement sont sauvées dans un script et peuvent être rechargée en cas de problème. De plus, un enregistrement de l'organisation de l'infrastructure dans des fichiers texte peut être demandé par l'administrateur. L'organisation de l'infrastructure peut donc être elle aussi sauvegardée. Cependant, en cas de panne du serveur hébergeant l'outil d'administration, seul un stockage de ces fichiers sur des serveurs distants peut éviter toute perte

d'information.

L'outil d'administration proposant un langage dédié, l'administrateur pourrait donc s'attendre à des fonctionnalités de programmation de politique d'ordonnancement à l'aide de relation « si... alors... ». Par exemple, si un serveur a une consommation CPU de 100 %, il faut migrer la VM consommant le plus de CPU sur le serveur consommant le moins de CPU. Cependant, la description d'une politique d'ordonnancement efficace comme la consolidation est difficile à décrire à l'aide de ce type de relations. Par exemple, la description de la migration d'une VM hébergée sur un serveur non surchargé afin d'éteindre le serveur n'est pas triviale.

La taille des infrastructures virtualisées étant de plus en plus grandes, les systèmes autonomes sont nécessaires pour surveiller et appliquer des politiques d'ordonnancement efficaces. De plus, avec l'informatique dans les nuages, l'administrateur n'a plus besoin de savoir l'emplacement exact d'une VM. Un système « intelligent » peut placer la VM et assurer sa demande en ressources. Cependant, l'administrateur doit pouvoir obtenir des informations sur la VM comme son propriétaire ou le but de son déploiement. L'administrateur doit aussi pouvoir reconfigurer les politiques d'ordonnancement mises en place et continuer d'administrer l'infrastructure en cas de panne du système autonome. L'outil d'administration choisi opte pour la désignation de l'administrateur comme seul acteur des reconfigurations de l'infrastructure. Un système totalement autonome n'est donc pas envisageable en ces termes même si l'administrateur ne peut se passer d'algorithmes d'ordonnancement pour calculer les reconfigurations complexes. L'outil d'administration présenté dans ce document propose une surveillance de l'infrastructure et un outil de calcul de plan de reconfigurations. Cependant, le plan de reconfigurations n'est pas assez précis pour permettre à l'administrateur de choisir l'exécution d'une partie du plan. Pour cela, des relations cause/conséquence entre les problèmes détectés et les reconfigurations doivent être établies. L'administrateur pourrait alors choisir d'appliquer les reconfigurations les plus importantes et/ou les plus pertinentes. Par exemple, l'administrateur pourrait décider d'appliquer les reconfigurations liées à la surcharge d'un serveur et de ne pas prendre en compte certaines migrations liées à l'exécution d'une politique de consolidation.

L'administration d'une infrastructure virtualisée de taille conséquente peut être prise en charge par plusieurs administrateurs. Par exemple, une infrastructure distribuée sur 3 sites peut être administrée par 3 administrateurs. Dans ce cas, l'administrateur responsable du site de Nantes ne pourra exécuter de reconfigurations sur le site de Paris. Cependant, il aura accès aux mécanismes d'introspection afin de pouvoir suivre l'évolution d'un projet distribué sur plusieurs sites. La gestion d'une même infrastructure par plusieurs utilisateurs nécessitent donc la mise en place de permissions sur diverses parties de l'infrastructure.

Chapitre 8

Annexes

8.1 Étalonnage d'un algorithme génétique

De nombreuses variables sont utilisées dans les algorithmes génétiques. L'importance de ces variables varient suivant le problème à résoudre. Afin de configuration les variables le plus précisément possibles, deux étalonnages ont été effectués : un étalonnage à gros grains et un étalonnage plus précis.

Étalonnage à gros grains L'étalonnage à gros grains sert à mesurer l'impact de chaque variable. Chaque variable est initialisée à sa valeur minimale. Ensuite, les valeurs sont augmentées, une à une, jusqu'à leur maximum afin de déterminer la plage de valeurs proches de l'optimum. La qualité de la solution donnée par l'algorithme est évaluée en fonction du nombre de problèmes non résolus et le coût du plan de reconfiguration, c'est-à-dire le nombre d'opérations nécessaires pour obtenir l'organisation voulue. La solution optimale ne comporte aucun problème et nécessite trois opérations pour être atteinte. Afin d'obtenir des valeurs cohérentes, cinq cent mesures sont effectuées et les relevés ci-dessous présentent les moyennes observées.

Les valeurs initiales des variables sont :

- le nombre de générations dépend du nombre de VM. La taille initiale est d'une fois le nombre de VM ;
- la taille la population dépend du nombre de VM. La taille initiale est de une fois le nombre de VM ;
- le nombre de reproductions est exprimé par un pourcentage de la population. La valeur initiale est de 50 % ;
- le nombre de mutations est exprimé par un pourcentage de la population. La valeur initiale est de 20 % de la population ;
- le nombre de gènes modifiés lors d'une mutation est exprimé par un pourcentage du nombre de VM. La valeur initiale est fixée à 10 %.

Taille de la population (x Nb de VM)	Problèmes	Coût
1	1,59	3,13
2	1,49	3,12
3	1,35	3,28
4	1,36	3,21
5	1,32	3,22

TABLE 8.1 – Influence de la taille de la population

Nombre de mutations (% de la population)	Problèmes	Coûts
20	1,59	3,17
40	1,53	3
60	1,5	2,91
80	1,5	2,94
100	1,51	2,82

TABLE 8.2 – Influence du nombre de mutations

Nombre de gènes modifiés (% du nombre de VM)	Problèmes	Coûts
10	1,56	3,22
30	1,03	3,53
50	0,89	3,73
70	0,76	4
90	0,64	4,29

TABLE 8.3 – Influence du nombre de gènes modifiés par mutation

Nombre de générations (x nombre de VM)	Problèmes	Coûts
1	1,56	3,2
2	1,53	3,08
3	1,55	3,04
4	1,58	3,03
5	1,53	3,08

TABLE 8.4 – Influence du nombre de générations

Étalonnage précis de l'algorithme Une fois l'impact des variables établies, la valeur optimale de chaque variable est calculé de façon plus précise. Afin d'évaluer la configuration courante des variables, le calcul est effectué cinq cent fois et le pourcentage de solutions correctes est évaluée à partir du pourcentage de solutions sans problèmes. Parmi ces solutions, celles pouvant être atteintes avec un plan de reconfiguration minimal, c'est-à-dire comprenant trois opérations,

Nombre de reproductions (% de la population)	Problèmes	Coûts
50	1,56	3,32
60	1,56	3,23
70	1,6	3,08
80	1,59	3,16
90	1,57	3,12

TABLE 8.5 – Influence du nombre de reproductions

sont différenciées de celles nécessitant un plan de reconfiguration comprenant quatre opérations.

Taille de la population (x nombre de VM)	Solutions sans problème (%)	Coût optimal (%)	Coût optimal +1 (%)
1	89	27	39
2	99	49	42
3	100	68	31
4	100	85	14
5	100	87	13

TABLE 8.6 – Affinage de la taille de la population

Nombre de mutation (% nombre de VM)	Solutions sans problème (%)	Coût optimal (%)	Coût optimal +1 (%)
20	96	32	45
40	100	53	39
60	100	72	25
80	100	78	20
100	100	84	15

TABLE 8.7 – Affinage du nombre de mutations

Nombre de gènes modifiés (% du nombre de VM)	Solutions sans problème (%)	Coût optimal (%)	Coût optimal +1 (%)
10	10	42	10
30	98	64	32
50	99	70	26
70	99	68	29
90	100	62	35

TABLE 8.8 – Affinage du nombre de gènes modifiés par individus

Nombre de génération (x nombre de VM)	Solutions sans problème (%)	Coût optimal (%)	Coût optimal +1 (%)
1	99	39	48
2	100	68	28
3	100	69	28
4	100	77	22
5	100	77	21

TABLE 8.9 – Affinage du nombre de générations

Nombre de reproduction (% de la population)	Solutions sans problème (%)	Coût optimal (%)	Coût optimal +1 (%)
20	100	55	40
30	100	63	34
40	100	66	32
50	100	72	25
60	100	69	29
70	100	72	25
80	99	67	31
90	99	66	30

TABLE 8.10 – Affinage du nombre de reproductions

8.2 Description d'une architecture à l'aide de VMDesc

L'infrastructure décrite à l'aide du langage VMDesc comprend 95 serveurs et 450 VM. Les serveurs sont répartis sur plusieurs grappes elles-mêmes réparties sur deux sites localisés à Grenoble et Nancy. Cette description peut être utilisée par l'outil de déploiement VGrid.

8.2.1 Description de l'organisation des serveurs

Le fichier suivant décrit la vue physique, c'est-à-dire l'organisation des serveurs dans l'infrastructure. Leur rôle définit leur fonction au sein de l'infrastructure. Les hyperviseurs hébergent des VM dont les images disque sont stockées sur les serveurs NFS (Network File System). La propriété « cmd » permet de choisir le script de configuration du serveur.

```
1  # Grenoble
2  site: grenoble
3  edel
4  genepi
5
6  cluster: edel
7  nfs1
8  host[1-14]
9
10 cluster: genepi
11 nfs2
12 host[15-25]
13
14 # Nancy
```

```
15 site: nancy
16 graphene
17 griffon
18
19 cluster: graphene
20 nfs4
21 host[26-65]
22
23 cluster: griffon
24 nfs3
25 host[66-95]
26
27 # Server classification
28 server: nfs[1-4]
29 model = nfs
30
31 server: host[1-95]
32 model = hypervisor
33
34 # Model descriptions
35 model: hypervisor
36 env = squeeze-x64-base
37 role = hypervisor
38 cmd = hypervisor.sh
39
40 model: nfs
41 env = squeeze-x64-base
42 role = nfs
43 cmd = nfs.sh
```

8.2.2 Description de l'organisation des machines virtuelles

Le fichier suivant décrit la vue logique, c'est-à-dire l'organisation des VM dans l'infrastructure. Les VM sont regroupées en vjob (virtual job) d'après leur système d'exploitation. Par exemple, le vjob « slitaz » regroupent 150 VM exécutant le système d'exploitation slitaz, un système Linux minimaliste. L'utilisation de ce système est utile pour créer des VM nécessitant très peu de ressources disque et mémoire.

```
1  vjob: debian
2  vm[1-150]
3
4  vjob:slitaz
5  vm[151-300]
6
7  vjob:ubuntu
8  vm[301-450]
9
10 vm: vm[1-150]
11 env = squeeze-min
12 mem = 600
13
14 vm: vm[151-300]
15 env = slitaz
16 mem = 200
17
18 vm: vm[301-450]
19 env = ubuntu
20 mem = 700
```

8.2.3 Description des relations d'hébergement entre les hyperviseurs et les machines virtuelles

Le fichier suivant décrit les ponts entre les deux vues. Toutes les VM doivent être affectée à un serveur. La description complète n'étant pas pertinente, seul le début du fichier est exposé.

```
# Generate from the script mapping.sh
host: host1
vm[1-4]

host: host2
vm[5-10]

host: host3
vm[11-14]

host: host4
vm[15-20]
```

8.3 Relevé des consommations CPU de machines virtuelles

8.3.1 Outils utilisés pour la gestion de la ressource CPU d'un hyperviseur

8.3.1.1 Code source en langage C du générateur de ressource CPU

```
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char* argv[]) {
        unsigned long temp,loop, max_loop = 0, loop_usec,burning,sleeping;
        // The experimentation time in seconds
        int period = atoi(argv[2]);
        // The load to create
        int load = atoi(argv[1]);
        // A start time
        time_t start,stop;
        // The elapsed time
        double elapsed = 0;
        if(argc != 3 && argc != 4) {
                printf("Wrong number of arguments\n");
                printf("\tparameter 1: load to apply\n");
                printf("\tparameter 2: the experimentation time\n");
                return 13;
        } else {
                // Time for burning cpu in microseconds
                burning = 10000 * load / 100;
                printf("Burning time: %lu\n",burning);
                // Time for sleeping in microseconds
                sleeping = 10000 - burning;
                printf("Sleeping time: %lu\n",sleeping);
                if(argc == 4) {
                        printf("Number of computation per usec: %d\n",atoi(argv[3]));
                        max_loop = burning * atoi(argv[3]);
                        printf("Number of computation per iteration: %lu\n",max_loop);
                } else {//argc == 3
                        // Compute max_loop from cpu benchmark
```

```
34                          while(elapsed < 3) {
35                                  max_loop += 500000000;
36                                  start = time(NULL);
37                                  for (loop = 0 ; loop < max_loop ; loop++)
38                                          sqrt(1234567890);
39                                  stop = time(NULL);
40                                  elapsed = difftime(stop,start);
41                          }
42                          // Number of computation for the burning time
43                          max_loop /= (elapsed * 1000000);
44                          printf("Number of computation per usec: %lu\n",max_loop);
45                          max_loop *= burning;
46                          printf("Number of computation per iteration: %lu\n",max_loop);
47                  }
48          }
49          // Here, the burning the process
50          start = time(NULL);
51          for(period *= 100; period > 0 ; period--) {
52                  for (loop = 0 ; loop < max_loop ; loop++)
53                          sqrt(1234567890);
54                  usleep(sleeping);
55          }
56          printf("The burning lasts %g\n",difftime(time(NULL),start));
57          return 0;
58  }
```

8.3.1.2 Script Bash de récupération de la charge CPU à partir de l'outil dstat

```
1  while [ true ]; do
2          echo "cpucons: $(dstat -c 1 1|tail -n 1|awk '{ print $1+$2 }') ($(date +%T))"
3          sleep 2
4  done
```

8.3.1.3 Code source en langage C de la récupération de la charge CPU à partir de l'hyperviseur

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <libvirt/libvirt.h>
4  #include <time.h>
5
6  #define TIME_SEC 2
7  #define NB_MAX_VMS 50
8  #define NB_HOST_CPU 2
9
10 int main(int argc, char* argv[]) {
11         int iteration,ite;
12         char uri[100];
13         char buffer[256];
14         int ids[NB_MAX_VMS],nbRunnings,i;
15         // Names of VM
16         char vms[NB_MAX_VMS][30];
17         int cpu[NB_MAX_VMS];
18         if(argc != 3 && argc != 2) {
19                 printf("Usage: cpucons name_of_hypervisor nb_of_iteration\n");
20                 return 13;
21         } else if(argc == 2) // Almost endless loop
22                 iteration = 20000;
```

```
23          else // argc == 3
24                  iteration = atoi(argv[2]);
25          // Connect to the hypervisor
26          sprintf(uri,"qemu+ssh://%s/system",argv[1]);
27          printf("Connection to %s\n",uri);
28          virConnectPtr host = virConnectOpen(uri);
29          // Get the id of the running VM
30          nbRunnings = virConnectListDomains(host,ids,NB_MAX_VMS);
31          for(ite = 0 ; ite < iteration ; ite++) {
32                  for(i = 0; i < nbRunnings ; i++) {
33                          virDomainPtr ptr = virDomainLookupByID(host,ids[i]);
34                          // Get the name of the VM
35                          printf("VM: %s, ",virDomainGetName(ptr));
36                          // Get the cpu utilization for the VM
37                          virVcpuInfo info[1];
38                          virDomainGetVcpus(ptr,info,1,NULL,0);
39                          /* Only test for one vcpu per VM */
40                          unsigned long long btiming = info[0].cpuTime;
41                          sleep(1);
42                          virDomainGetVcpus(ptr,info,1,NULL,0);
43                          time_t timestamp = time(NULL);
44                          strftime(buffer, sizeof(buffer), "%X", localtime(&timestamp));
45                          printf("cpucons: %llu (%s)\n", \
46                                  (info[0].cpuTime - btiming) / (10000000 * NB_HOST_CPU),buffer);
47                  }
48                  sleep(TIME_SEC);
49          }
50          return 0;
51  }
52
```

8.3.2 Relevé des consommations CPU

Consommations CPU observées à partir de l'hyperviseur KVM						
temps (s)	vm1	vm2	vm3	vm4	vm5	vm6
0	7	8	13	13	18	22
8	11	7	13	17	22	21
16	7	8	13	13	14	20
24	11	7	13	18	18	24
32	8	8	13	13	14	18
40	11	7	13	14	18	24
48	7	12	13	13	14	20
56	7	7	13	14	13	20
64	7	12	13	18	14	18
72	7	8	13	14	13	18
80	7	12	13	19	16	23
moyenne (%)	8	9	15	19	16	21

Consommations CPU observées à l'aide de l'outil dstat						
temps (s)	vm1	vm2	vm3	vm4	vm5	vm6
0	15	14	32	31	59	62
8	12	15	31	25	64	61
16	14	13	31	30	62	61
24	15	14	35	29	61	59
32	14	14	32	31	64	63
40	13	15	33	31	63	60
48	13	15	33	27	63	61
56	12	13	30	28	62	62
64	14	15	33	26	61	64
72	12	14	32	27	59	64
80	12	13	31	31	61	60
moyenne (%)	13	14	32	29	62	62

8.4 Les propriétés de configuration de VMScript

Les propriétés suivantes configurent l'ensemble des modules de VMScript.

Nom de la propriété	configuration obtenue	valeurs disponibles	valeur par défaut
executor.action	nombre maximum d'actions exécutées en parallèle	-	20
guardian.period.second	Temps entre deux analyses du module de surveillance	-	10
libvirt.uri	URI utilisée pour la connexion à libvirt	-	qemu+ssh ://NAME/system
monitor.period.second	Temps entre deux analyses du module d'observation	-	10
placement.algorithm	Choix du module de placement	Genetic, Entropy	Genetic
placement.auto	Demande de confirmation avant application du plan de reconfiguration	true, false	false
simulate.power	Simulation la consommation électrique en fonction de la consommation CPU	true, false	false
timer.period.second	Temps entre deux analyses du gestionnaire de temps	-	60
view.mapping	Définition des connexions entre les vues	-	server/vm
view.view_name.types	Définiton des types d'une vue	-	-
vmflow.desc	Fichier de description d'infrastructure utilisé par le simulateur de charge	-	vmflow/desc.vmflow
vmflow.flow	Fichier de description de charge utilisé par le simulateur de charge	-	vmflow/flow.vmflow
vmscript.agents	Liste des modules de reconfiguration	-	localhost :3131
vmscript.backend	Type des modules internes d'observation et de reconfiguration à charger		""
vmscript.descriptions	le répertoire contenant les fichiers de description	-	descriptions
vmscript.file	Nom des fichiers de descriptions à charger	-	""
vmscript.fs	Séparateur de champs pour les communications TCP	-	+
vmscript.monitors	Liste des modules d'observation	-	localhost :3132
vmscript.rs	Séparateur d'enregistrements pour les communications TCP	-	%

Nom de la propriété	configuration obtenue	valeurs disponibles	valeur par défaut
vmscript.scripts	le répertoire contenant les fichiers de scripts	-	scripts
vmscript.server.port	Port utilisé par le serveur VMScript	-	8182
vmscript.type.misc	Type lié à la classe représentant les éléments « misc »	-	misc
vmscript.type.server	Type lié à la classe représentant les éléments « server »	-	server
vmscript.type.vm	Type lié à la classe représentant les éléments « vm »	-	vm
vmscript.view	Les vues de VMScript	-	physical, logical

Bibliographie

[ACJ08] Heithem Abbes, Christophe Cérin, and Mohamed Jemni. Pastrygrid : decentralisation of the execution of distributed applications in desktop grid. In *Proceedings of the 6th international workshop on Middleware for grid computing*, MGC '08, pages 4 :1–4 :6, New York, NY, USA, 2008. ACM.

[All09] Globus Alliance. Extended resource specification language (xrsl). Technical report, Globus Alliance, 2009.

[Ama] Amazon elastic compute cloud. http ://aws.amazon.com/ec2/.

[Anj05] Ali Anjomshoaa. Job submission description language (jsdl) specification, version 1.0. Technical report, Global Grid Forum, 2005.

[AOVP05] Jeannie Albrecht, David Oppenheimer, Amin Vahdat, and David A. Patterson. Design and implementation tradeoffs for wide-area resource discovery. In *In Proceedings of 14th IEEE Symposium on High Performance, Research Triangle Park*, pages 113–124. IEEE Computer Society, 2005.

[BDF+03] 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.

[BKB07] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 119 –128, 21 2007-yearly 25 2007.

[BL94] R.D. Blumofe and C.E. Leiserson. Scheduling multithreaded computations by work stealing. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 356 –368, nov 1994.

[BM02] Luciano Porto Barreto and Gilles Muller. Bossa : a language-based approach to the design of real-time schedulers. In *10th International Conference on Real-Time Systems (RTS'2002)*, pages 19–31, Paris, France, March 2002.

[BR10] David J. Brown and Charles Reams. Toward energy-efficient computing. *Commun. ACM*, 53 :50–58, March 2010.

[BSDCP10] Damien Borgetto, Patricia Stolf, Georges Da Costa, and Jean-Marc Pierson. Energy aware autonomic managers. In *ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy), Passau, Germany, 13/04/2010-15/04/2010*, http ://www.acm.org/, avril 2010. ACM.

[CA07] Alexandru Caracas and Jörn Altmann. A pricing information service for grid computing. In *Proceedings of the 5th international workshop on Middleware for grid computing : held at the ACM/IFIP/USENIX 8th International Middleware Conference*, MGC '07, pages 4 :1–4 :6, New York, NY, USA, 2007. ACM.

[CA11] J. Celaya and U. Arronategui. A highly scalable decentralized scheduler of tasks with deadlines. In *Grid Computing (GRID), 2011 12th IEEE/ACM International Conference on*, pages 58 –65, sept. 2011.

[CBC+08] Benoît Combemale, Laurent Broto, Xavier Crégut, Michel J. Daydé, and Daniel Hagimont. Autonomic management policy specification : From uml to dsml. In *MoDELS*, pages 584–599, 2008.

[CCsK+04] Andrew Chien, Henri Casanova, Yang suk Kee, , and Richard Huang. The virtual grid description language : vgdl. Technical report, VGrADS Project, 2004.

[CDCG+05] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard. A batch scheduler with high level components. In *CCGRID '05 : Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 776–783, Washington, DC, USA, 2005. IEEE Computer Society. OAR.

[CFH+05] 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.

[CGH09] Damien Cerbelaud, Shishir Garg, and Jeremy Huylebroeck. Opening the clouds : qualitative overview of the state-of-the-art open source vm-based cloud management platforms. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, pages 22 :1–22 :8, New York, NY, USA, 2009. Springer-Verlag New York, Inc.

[CYC+01] Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. An empirical study of operating systems errors. *SIGOPS Oper. Syst. Rev.*, 35 :73–88, October 2001.

[DKL+08] Rajarshi Das, Jeffrey O. Kephart, Charles Lefurgy, Gerald Tesauro, David W. Levine, and Hoi Chan. Autonomic multi-agent management of power and performance in data centers. In *AAMAS '08 : Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 107–114, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[DLLC09] Pierre-Charles David, Thomas Ledoux, Marc Léger, and Thierry Coupaye. Fpath & fscript : Language support for navigation and reliable reconfiguration fractal architectures. *Annals of Telecommunications - Special issue on Software Components - The Fractal Initiative*, 64 :45–63, 2009.

[FDF03] Renato J. Figueiredo, Peter A. Dinda, and José A. B. Fortes. A case for grid computing on virtual machines. In *ICDCS '03 : Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 550, Washington, DC, USA, 2003. IEEE Computer Society.

[FK96] Ian Foster and Carl Kesselman. Globus : A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11 :115–128, 1996.

[FS08] Liana Fong and Malgorzata Steinder. Duality of virtualization : simplification and complexity. *SIGOPS Oper. Syst. Rev.*, 42 :96–97, January 2008.

[GAW09] Ajay Gulati, Irfan Ahmad, and Carl A. Waldspurger. Parda : proportional allocation of resources for distributed storage access. In *Proccedings of the 7th conference on File and storage technologies*, pages 85–98, Berkeley, CA, USA, 2009. USENIX Association.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[Gol73] 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.

[GPKC09] Pascale Vicat-Blanc Primet Guilherme Piegas Koslovski and Andrea Schwertner Charão. Vxdl : Virtual resources and interconnection networks description language. In *Networks for Grid Applications*, 2009.

[HLAP06] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K. Panda. A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 125–134, New York, NY, USA, 2006. ACM.

[HLM10] Fabien Hermenier, Adrien Lèbre, and Jean-Marc Menaud. Cluster-wide context switch of virtualized jobs. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 658–666, New York, NY, USA, 2010. ACM.

[HLMM11] Fabien Hermenier, Julia Lawall, Jean-Marc Menaud, and Gilles Muller. Dynamic consolidation of highly available web applications. Research Report RR-7545, INRIA, 02 2011.

[Inc06] IBM Inc. An architectural blueprint for autonomic computing. *Quality*, 36(June) :34, 2006.

[Jon01] Keith J. Jones. Loadable kernel modules. *Login*, 26 :42–49, November 2001.

[Kan06] Luke Kanies. Puppet : Next-generation configuration management. *j-LOGIN*, 31(1) :XX, February 2006.

[KCD+07] Jeffrey O. Kephart, Hoi Chan, Rajarshi Das, David W. Levine, Gerald Tesauro, Freeman Rawson, and Charles Lefurgy. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *ICAC '07 : Proceedings of the Fourth International Conference on Autonomic Computing*, page 24, Washington, DC, USA, 2007. IEEE Computer Society.

[KK09] Krishna and Kant. Data center evolution : A tutorial on state of the art, issues, and challenges. *Computer Networks*, 53(17) :2939 – 2965, 2009. <ce :title>Virtualized Data Centers</ce :title>.

[KKL+07] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm : the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, June 2007.

[Kot10] Lars Kotthoff. Constraint solvers : An empirical evaluation of design decisions. *CoRR*, abs/1002.0134 :XX, 2010.

[Koz90] John R. Koza. Genetic programming : a paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Stanford University, Stanford, CA, USA, 1990.

[KTK+09] Sanjay Kumar, Vanish Talwar, Vibhore Kumar, Parthasarathy Ranganathan, and Karsten Schwan. vmanage : loosely coupled platform and virtualization management in data centers. In *ICAC '09 : Proceedings of the 6th international conference on Autonomic computing*, pages 127–136, New York, NY, USA, 2009. ACM.

[Liba] Grid 5000. https ://www.grid5000.fr.

[Libb] La syntaxe de crontab. http ://fr.wikipedia.org/wiki/Crontab.

[Libc] libvirt : The virtualization api. http ://libvirt.org.

[Lib10] La consommation électrique des centres de données dans le collimateur, 05 2010. http ://www.bulletins-electroniques.com/actualites/63618.htm.

[Lie95] J. Liedtke. On micro-kernel construction. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, SOSP '95, pages 237–250, New York, NY, USA, 1995. ACM.

[LKK99] William Leinberger, George Karypis, and Vipin Kumar. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *Proceedings of the 1999 International Conference on Parallel Processing*, ICPP '99, pages 404–, Washington, DC, USA, 1999. IEEE Computer Society.

[MGVV07] Marvin McNett, Diwaker Gupta, Amin Vahdat, and Geoffrey M. Voelker. Usher : an extensible framework for managing custers of virtual machines. In *LISA'07 : Proceedings of the 21st conference on Large Installation System Administration Conference*, pages 1–15, Berkeley, CA, USA, 2007. USENIX Association.

[MK07] Dan Marinescu and Reinhold Kroeger. State of the art in autonomic computing and virtualization. Technical report, Distributed Systems Lab, Wiesbaden University of Applied Sciences, September 2007.

[NS07] Ripal Nathuji and Karsten Schwan. Virtualpower : coordinated power management in virtualized enterprise systems. In *SOSP '07 : Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 265–278, New York, NY, USA, 2007. ACM.

[NWG+09] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[PGW03] Rosario M. Piro, Andrea Guarise, and Albert Werbrouck. An economy-based accounting infrastructure for the datagrid. In *Proceedings of the 4th International Workshop on Grid Computing*, GRID '03, pages 202–, Washington, DC, USA, 2003. IEEE Computer Society.

[PLM10] Rémy Pottier, Marc Léger, and Jean-Marc Menaud. A reconfiguration language for virtualized grid infrastructures. In *10th IFIP international conference on Distributed Applications and Interoperable Systems (DAIS)*, volume 6115, pages 42–55, June 2010.

[PM12] Rémy Pottier and Jean-Marc Menaud. A safe management system for virtualized data center. In *The Eighth International Conference on Autonomic and Autonomous Systems (ICAS)*, March 2012.

[QL11a] Flavien Quesnel and Adrien Lebre. Operating systems and virtualization frameworks : From local to distributed similarities. In Yiannis Cotronis, Marco Danelutto, and George A. Papadopoulos, editors, *PDP '11 : Proceedings of the 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, pages 495–502, Los Alamitos, CA, USA, February 2011. IEEE Computer Society.

[QL11b] Flavien Quesnel and Adrien Lèbre. Cooperative dynamic scheduling of virtual machines in distributed systems. In *VHPC '11 : Proceedings of the 6th Workshop on Virtualization in High-Performance Cloud Computing*, page XX, Bordeaux, France, 2011.

[RC10] Jonathan Rouzaud-Cornabas. A distributed and collaborative dynamic load balancer for virtual machine. In *5th Workshop on Virtualization in High-Performance Cloud Computing (VHPC '10)*, page _, Ischia, Naples, Italy, August 2010. 10 pages.

[RG05] Mendel Rosenblum and Tal Garfinkel. Virtual machine monitors : Current technology and future trends. *Computer*, 38(5) :39–47, 2005.

[RRT+08] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "power" struggles : coordinated multi-level power management for the data center. In *ASPLOS XIII : Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 48–59, New York, NY, USA, 2008. ACM.

[SA10] Vijayaraghavan Soundararajan and Jennifer M. Anderson. The impact of management operations on the virtualized datacenter. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 326–337, New York, NY, USA, 2010. ACM.

[SABL06] Michael M. Swift, Muthukaruppan Annamalai, Brian N. Bershad, and Henry M. Levy. Recovering device drivers. *ACM Trans. Comput. Syst.*, 24 :333–360, November 2006.

[SCKN07] Renaud Sirdey, Jacques Carlier, Hervé Kerivin, and Dritan Nace. On a resource-constrained scheduling problem with application to distributed systems reconfiguration. *European Journal of Operational Research*, 183(2) :546 – 563, 2007.

[SG10] Vijayaraghavan Soundararajan and Kinshuk Govil. Challenges in building scalable virtualized datacenter management. *SIGOPS Oper. Syst. Rev.*, 44 :95–102, December 2010.

[SMLF09] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. An open source solution for virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5) :14–22, 2009.

[SN05] James E. Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5) :32–38, 2005. The ISA (Instruction Set Architecture).

[SRC10] J. Shafer, S. Rixner, and A.L. Cox. The hadoop distributed filesystem : Balancing portability and performance. In *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pages 122 –133, march 2010.

[SSVC10] Mark Stillwell, David Schanzenbach, Frédéric Vivien, and Henri Casanova. Resource allocation algorithms for virtualized service hosting platforms. *J. Parallel Distrib. Comput.*, 70 :962–974, September 2010.

[TDC+07] Gerald Tesauro, Rajarshi Das, Hoi Chan, Jeffrey O. Kephart, David Levine, Freeman L. Rawson III, and Charles Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. In *NIPS*, 2007.

[TRFD10] K. Tsakalozos, M. Roussopoulos, V. Floros, and A. Delis. Nefeli : Hint-based execution of workloads in clouds. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 74 –85, june 2010.

[TWML01] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor - a distributed job scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

[TWML02] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor : a distributed job scheduler. In *Condor : a distributed job scheduler*, 2002.

[VDKFLS10] Roman Van Der Krogt, Jacob Feldman, James Little, and David Stynes. An integrated business rules and constraints approach to data centre capacity management. In *Proceedings of the 16th international conference on Principles and practice of constraint programming*, CP'10, pages 568–582, Berlin, Heidelberg, 2010. Springer-Verlag.

[vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages : An annotated bibliography. *ACM-SIGPLAN Notices*, 35(6) :26–36, June 2000.

[vir] Virt-manager. http ://virt-manager.org/.

[vL12] Arnim van Lieshout. Managing vmware drs rules using powercli, 2012. http ://www.van-lieshout.com/2011/06/drs-rules/.

[VMwa] Vmware. http ://www.vmware.com.

[VMwb] Vmware vsphere. http ://www.vmware.com/fr/products/datacenter-virtualization/vsphere/.

[VMW06] VMWare. Vmware infrastructure : Resource management with vmware drs. Technical report, VMWare, 2006.

[VNS07] Geoffroy Vallee, Thomas Naughton, and Stephen L. Scott. System management software for virtual environments. In *CF '07 : Proceedings of the 4th international conference on Computing frontiers*, pages 153–160, New York, NY, USA, 2007. ACM. oscar-V.

[Wal02] Carl A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36 :181–194, December 2002.

[WSVY09] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper : Black-box and gray-box resource management for virtual machines. *Comput. Netw.*, 53 :2923–2938, December 2009.

[XZF+08] Jing Xu, Ming Zhao, José Fortes, Robert Carpenter, and Mazin Yousif. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing*, 11 :213–227, 2008. 10.1007/s10586-008-0060-0.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information Control*, 8 :338–353, 1965.

Glossaire

Administration Ensemble des tâches de gestion physique (ajout de composant informatique) et logicielle (installation d'application) d'un parc de serveurs.

AFS De l'anglais, « Andrew File System », soit littéralement « Système de fichier d'Andrew ». Ce système de fichier est inspiré de NFS. Il fut créé par Andrew Carnegie et Andrew Mellon..

API Acronyme de l'anglais, « Application Programming Interface ». Les API offrent l'utilisation de toutes ou d'une partie des fonctionnalités d'une application via une interface de programmation@.

Application Aussi appelée programme ou logiciel. Une application est un ensemble de programmes qui permet à un ordinateur ou à un système informatique d'assurer une tâche ou une fonction en particulier.

Application N-tiers Une application N-tiers est un programme ou un logiciel séparé en plusieurs parties appelées « tiers ». Le plus souvent, chaque tiers est distribué sur un serveur différent. Par exemple, l'application N-tiers la plus simple est l'architecture « client-serveur ». Un tiers nommé « serveur » met à disposition du contenu que le second tiers nommé « client » peut consulter.

Bash Acronyme pour « Bourne-Again SHell ». Bash est un interpréteur de lignes de commande dédié à l'administration de systèmes Unix.

CPU Acronyme de l'anglais, « Central Processing Unit ». Le CPU désigne la ressource processeur d'un serveur.

DaaS Acronyme de l'anglais, « Desktop as a Service » ou « Data as a Service ».

DHCP Acronyme de l'anglais Dynamic Host Configuration Protocol, littéralement, protocole de configuration dynamique d'un hôte. DHCP est un protocole réseau dont le rôle est d'assurer la configuration automatique des adresses IP de serveurs et/ou de machines virtuelles.

DRS Acronyme de l'anglais, « Dynamic Resource Scheduler ». Littéralement, Ordonnanceur de ressources dynamiques. Pour VMware, DRS représente le module de vSphere prenant en charge l'ordonnancement autonome des VM.

DSL Acronyme de l'anglais, « Domain Specific Language ». Littéralement, un langage dédié à un domaine spécifique.

Effet ping pong L'effet ping pong est obtenue lorsqu'une machine virtuelle est migrée d'un serveur A à un serveur B puis migrée de nouveau du serveur B au serveur A. Cet effet est un problème courant obtenu lors d'un ordonnancement de machines virtuelles basé sur les ressources consommées à un instant t. Par exemple, une machine virtuelle consommant

beaucoup de ressources va être migrée afin d'obtenir d'être seule sur un serveur et donc d'utiliser toutes les ressources disponibles. Si une politique de consolidation est mise en place, la machine virtuelle sera de nouveau migrer dès que sa consommation diminuera. Si sa consommation est très irrégulière la machine virtuelle sera constamment déplacée et l'effet ping pong apparaîtra.

Grappe une grappe est un regroupement de serveurs au sein d'un centre de données.

grid5000 Grid5000 est une plateforme expérimentale utilisée pour effectuer des études de systèmes distribués à grande échelle dans le monde de la recherche.

HTML De l'anglais « HyperText Markup Language », littéralement, langage de balisage pour l'hypertexte. Le langage HTML est un langage dédié à l'écriture de pages web.

Hyperviseur application nécessaire à la création et à la gestion de machines virtuelles.

IaaS Acronyme de l'anglais, « Infrastructure as a Service ».

Implémentation L'implémentation représente la phase où l'informaticien crée une application, notamment, à partir d'un langage de programmation.

Infrastructure Fédération de centres de données appartenant à une même entité.

Instanciation Le terme « instanciation » désigne la phase de création d'une VM. Toutes les ressources statiques de la VM sont crées comme son image disque ou la configuration de sa mémoire. Une fois l'instanciation terminée, la VM est prête à démarrer.

IP Acronyme de l'anglais, « Internet Protocole ». L'adresse IP permet d'identifier de façon unique un serveur sur un réseau.

JNA Acronyme de l'anglais, « Java Native Access ». La technologie JNA permet l'appel de codes écrits en langage C à partir du langage Java.

Linux Linux est le système d'exploitation libre le plus utilisé. Son noyau a été conçu par Linus Torvalds en 1992.

MATLAB De l'anglais « MATrix LABoratory », littéralement, laboratoire de matrices. Le langage MATLAB est un langage dédié aux calculs scientifiques.

Migration à chaud Déplacement d'une machine virtuelle d'un serveur à un autre sans arrêter la machine virtuelle.

Minitel Le Minitel était utilisé en France dans les 1980 pour utiliser le service Vidéotex. Le service le plus utilisé était l'annuaire téléphonique accessible à partir du numéro 3611. Le service a été progressivement abandonné au profit d'Internet. La fermeture définitive aura lieu le 30 juin 2012.

NAT Acronyme de l'anglais, « Network Address Translation ». Le mode NAT est utilisé par l'hyperviseur pour faire communiquer une VM avec un réseau externe. Cependant, la VM n'est pas visible à partir du réseau.

NFS De l'anglais, « Network File System », soit littéralement « Système de fichiers en réseau ». Ce système de fichiers partagé a été créé par Sun Microsytems pour relier des systèmes UNIX. Aujourd'hui, la plupart des systèmes d'exploitation le prennent en charge..

Open Source Le terme « Open Source » peut être traduit par « code source libre ». Le code source, écrit à l'aide d'un langage de programmation, permet de construire une application. Le code source d'une application « Open Source » est accessible afin de permettre la modification de cette application par n'importe quel développeur, notamment, afin de corriger les bogues.

OSGI Acronyme de l'anglais, « Open Service Gateway Initiative ». La technologie OSGI permet de contruire une application, programmée à l'aide du langage Java, à partir composants petits, réutilisables et collaboratifs.

PaaS Acronyme de l'anglais, « Platform as a Service ».

PG Acronyme pour « Programmation Génétique ».

Politique d'ordonnancement Politique décrivant l'organisation des machines virtuelles sur l'ensemble de l'infrastructure.

PPC Acronyme pour « Programmation Par Contraintes ».

PUE Acronyme de l'anglais, « Power Usage Effectiveness ». Le PUE représente le rendement énergétique d'un centre de données. C'est le rapport entre l'électricité utilisé pour faire tourner les serveurs et l'électricité totale consommées par le centre de données.

RMI Acronyme de l'anglais, « Remote Method Invocation ». La technologie RMI permet l'appel distant de méthodes entre applications Java.

SaaS Acronyme de l'anglais, « Software as a Service ».

Script Un script est une séquence de commandes devant être interprétées et exécutées dans un ordre précis.

SDK De l'anglais « Software Development Kit », littéralement, kit de développement d'application. Un SDK est une librairie permettant un développement plus facile d'application utilisant une technologie précise. Par exemple, le Java SDK permet le développement d'applications Java.

Serveur Un serveur est un ordinateur prévu pour être hébergé dans un centre de données. Les dimensions des serveurs sont standardisées afin d'empiler les serveurs dans des armoires. Les serveurs sont aussi plus performants et plus fiables que les ordinateurs de bureau.

Sirocco Sirocco est un gestionnaire de machines virtuelles Open Source permettant la gestion des hyperviseurs Hyper-V, VMWare VCenter, Kvm et Xen.

SLA Acronyme de l'anglais, « Service Level Agreement ». Le SLA définit la qualité de service requise entre un prestataire et un client.

Socket Une socket permet d'établir une connexion TCP/IP entre deux applications.

SQL De l'anglais « Structured Query Language », littéralement, language de requêtes structurées. Le langage SQL est un langage dédié à la base de données.

SSH Acronyme de l'anglais « Secure Shell ». Ssh est à la fois un protocole de communication et une application d'accès à distance sécurisé basée sur le chiffrement des communications à l'aide de clés..

Système d'exploitation Le système d'exploitation est la couche logicielle permettant l'utilisation des ressources matérielles. Sans lui, un serveur n'est qu'un ensemble de composants électroniques.

TCP Acronyme de l'anglais, « Transfert Control Protocol ». TCP est un protocole de transport de données fiable utilisé lors de communications réseaux.

UNIX Un système UNIX est un système d'exploitation multitâche et multi-utilisateur créé en 1969 par Kenneth Thompson..

URL Acronyme de l'anglais, « Uniform Resource Locator », littéralement, « Localisateur Uniforme de Ressource ». Une url est une adresse Web, c'est-à-dire, une chaîne de caractères utilisée pour adresser une ressource de l'Internet.

VCPU Acronyme de l'anglais, « Virtual Central Processing Unit ». Le VCPU désigne la ressource processeur d'une machine virtuelle.

VM Acronyme de l'anglais, « Virtual Machine », littéralement machine virtuelle.

VMM Acronyme de l'anglais, « Virtual Machine Manager ». Littéralement, un gestionnaire de machines virtuelles. Le VMM fournit les opérations de gestion de machines virtuelles comme le démarrage et la migration.

VMS Acronyme de l'anglais, « Virtual Machine Scheduler ». Littéralement, un ordonnanceur de machines virtuelles. Le VMS est un outil autonome responsable de l'organisation des machines virtuelles sur l'infrastructure.

WSDL Acronyme de l'anglais, « Web Service Description Language ». La technologie WSDL décrit une interface de communication dans le but d'utiliser un service web.

XML Acronyme de l'anglais, « Extensible Markup Language », soit littéralement « langage de balisage extensible ». Ce langage se caractérise par la présence de balise permettant des contenus complexes sous formes d'arbres. Tout document XML peut être défini et validé par un schéma XML..

Rémy Pottier

VMScript, un langage dédié à l'administration d'infrastructures virtualisées
VMScript, A Domain Specific Language For Virtualized Infrastructures

Résumé

Avec l'émergence de l'informatique dans les nuages, la capacité d'hébergement des centres de données ne cesse d'augmenter afin de répondre à une demande de plus en plus forte. La gestion, appelée l'administration, d'un centre de données entraîne des opérations fréquentes sur des machines virtuelles (VM) ainsi que sur des serveurs. De plus, chaque VM hébergée possède des besoins spécifiques au regard de sa qualité de service, de ses ressources et de son placement qui doit être compatible avec les mécanismes de tolérance aux pannes et la configuration ré-seau. Les outils de « l'Infrastructure As A Service » tels que OpenNebula et Vmware vSphere simplifient la création et le déploiement de VM. Cependant, l'administration d'une infrastruc-ture virtualisée repose encore sur des changements manuels décidés par les administrateurs. Cette approche n'est plus pertinente pour la gestion d'infrastructures virtualisées de milliers de VM. En effet, les administrateurs ne peuvent pas manipuler des ensembles importants de VM tout en assurant la compatibilité des reconfigurations exécutées avec les besoins des VM. De nouvelles approches d'administration d'infrastructures proposent l'automatisation de certaines tâches d'administration. L'outil décrit dans ce document utilise des langages dédiés pour ré-pondre aux besoins d'administration infrastructures virtualisées de taille conséquente. Dans un premier temps, l'outil propose aux administrateurs des opérations d'introspection pour obser-ver l'organisation des ressources déployées sur l'infrastructure et les reconfigurations habituelles comme le démarrage, l'arrêt et le redémarrage de VM et de serveurs. Dans un second temps les administrateurs définissent le placement des VM à partir de règles de placement. À partir de ces règles, l'outil d'administration vérifie chaque reconfiguration et chaque ajout de règles exécutés par l'administrateur. Si une reconfiguration ou une règle est invalide, l'outil détecte un conflit et avertit l'administrateur de l'échec de l'opération. L'outil d'administration, à l'aide d'algorithmes d'ordonnancement peut calculer un plan de reconfigurations résolvant les conflits. Ces algorithmes peuvent aussi être utilisés pour mettre en place des politiques d'ordonnancement comme la consolidation ou l'équilibrage de charge.

Mots clés

machine virtuelle, langage dédié, administration

Abstract

With the emergence of cloud computing, the hosting capacity of the data centers has been continuously growing to support the non-stop increasing clients demand. Managing a data center implies to regularly manipulate both virtual machines (VM) and servers. Each hosted VM has specific expectations regarding its quality of service, its resource requirements and its placement that may be compatible with fault tolerance mechanisms and the networking configuration. Infra-structure As A Service solutions such as OpenNebula and VMWare vSphere extremely simplify creations and deployments of VM but virtualized infrastructure management is still relying on manual changes on the environment. This approach is no longer compatible with an infrastruc-ture composed of thousand of VM. Indeed, a system administrator can not manipulate a large set of VM insuring that its reconfigurations are compatible with the expected VM requirements. This situation has led to new approaches for the infrastructure management employing auto-mation to replace the traditional manual approach. The tool described in this document deals with VM management from Domain Specific Languages. On the one hand, this tool proposes to administrators introspection operations to monitor the infrastructure resources and common reconfigurations including starting, halting, rebooting, of servers and VM. On the other hand, administrators define the VM placement from placement rules. Then, the system checks, accor-ding to active rules, the validity of all reconfigurations and rules performed by administrators. If a reconfiguration or a rule is invalid, the administrative tool detects conflicts and warns adminis-trators. To resolve a conflict, the system, by interacting with scheduling algorithms, computes a reconfiguration plan that satisfies all rules. The reconfiguration plan can also apply scheduling policies as consolidation or load balancing with respect to placement rules.

Keywords

Virtual Machine, Domain Specific Language, Management

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

ECOLE DES MINES DE NANTES