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

ECOLE DES MINES DE NANTES

Thèse de Doctorat

Flavien QUESNEL

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

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27
Spécialité : Informatique
Unité de recherche : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 20 février 2013
Thèse n° : 2013EMNA0046

Vers une gestion coopérative des infrastructures virtualisées à large échelle : le cas de l'ordonnancement

JURY

Rapporteurs :	**M. Frédéric DESPREZ**, Directeur de recherche, INRIA Rhône-Alpes **M. Daniel HAGIMONT**, Professeur, ENSEEIHT Toulouse
Examinateurs :	**M^me^ Christine MORIN**, Directrice de recherche, INRIA Rennes **M. Jean-Marc MENAUD**, Maître-assistant, École des Mines de Nantes **M. Patrick MARTINEAU**, Professeur des Universités, Polytech Tours **M. Sébastien GOASGUEN**, Community Manager CloudStack EMEA, Citrix
Directeur de thèse :	**M. Mario SÜDHOLT**, Professeur, École des Mines de Nantes
Co-directeur de thèse :	**M. Adrien LÈBRE**, Chargé de recherche, École des Mines de Nantes

Remerciements

Ces années de doctorat ont été pour moi l'occasion de rencontrer, travailler et interagir avec un grand nombre de personnes, que je souhaite maintenant remercier.

Je remercie tout d'abord les membres de mon jury, pour avoir accepté d'évaluer mes travaux ; en particulier, merci à :

- Frédéric DESPREZ, pour avoir accepté la fonction de rapporteur et pour la collaboration qui a permis d'évaluer DVMS sur la plate-forme Grid'5000 ;
- Daniel HAGIMONT, pour avoir accepté d'être rapporteur et pour avoir suivi mes travaux ;
- Christine MORIN, pour avoir suivi ma soutenance à une heure très matinale et pour les discussions très constructives que nous avons eues dans le cadre du suivi de thèse ;
- Jean-Marc MENAUD, pour avoir obtenu le financement qui m'a permis d'effectuer ce doctorat ;
- Patrick MARTINEAU, pour avoir présidé le jury et assuré l'expression de tous ses membres ;
- Sébastien GOASGUEN, pour avoir suivi ma soutenance alors que vous étiez en déplacement ;
- Mario SÜDHOLT, pour m'avoir offert la possibilité de poursuivre un doctorat, pour toutes les discussions constructives que nous avons eues afin d'améliorer DVMS, et pour avoir vérifié formellement l'absence d'interblocages dans DVMS ;
- Adrien LÈBRE, pour m'avoir proposé ce sujet de doctorat et pour m'avoir guidé dans son étude ; merci pour ta passion de la recherche, pour le temps que tu m'as consacré, pour ta disponibilité, ton aide, ton empathie et ta sympathie, ta patience, les longues discussions que nous avons eues pour donner naissance à DVMS et l'améliorer, les nuits voire les weekends passés à rédiger des articles, ainsi que tes nombreuses remarques et corrections qui ont permis d'améliorer de manière significative la qualité de ce manuscrit et de la présentation pour la soutenance.

Merci à ceux dont j'ai partagé le bureau, pour l'excellente ambiance qui y régnait et pour tout ce que nous avons vécu et partagé ensemble. Je vous souhaite tout ce qu'il y a de meilleur pour la suite. En particulier, merci à (par ordre chronologique d'arrivée et de départ) :

- Marc LÉGER, pour nos échanges sur nos études d'ingénieur respectives et sur les concepts relatifs aux systèmes d'exploitation et aux hyperviseurs ;
- Ismael MEJÍA, pour nos grandes discussions sur les mondes académique et industriel, ainsi que sur la gastronomie en France et ailleurs ;
- Rémy POTTIER, pour toute l'aide que tu m'as apportée, pour les anecdotes qui ont émaillé ton trajet pour te rendre à l'école Grid'5000 en 2010 et qui me font toujours sourire quand j'y repense, et pour toutes les discussions sur les systèmes d'exploitation, la virtualisation, l'informatique en nuage et l'administration système (ce qui m'a d'ailleurs permis de découvrir la distribution Slitaz) ;
- Frederico ÁLVARES, pour nos travaux communs autour de l'évaluation de la consommation énergétique des machines virtuelles, pour nos débats sur la langue française (qui m'ont d'ailleurs permis de réviser la grammaire en

profondeur), et pour toutes les conversations du quotidien ;
– Gustavo BRAND, pour nos discussions sur les composants informatiques et la photographie ; profite bien des moments que tu passes avec ton petiot :-) ;
– Jonathan PASTOR, pour ton travail acharné qui a permis de valider DVMS avec SimGrid, et pour la magnifique affiche que tu as réalisée pour promouvoir ma soutenance.

Merci aux autres personnes avec lesquelles j'ai longuement travaillé :
– Daniel BALOUEK, pour avoir développé Flauncher, et pour toutes les nuits que nous avons passées à le déployer sur Grid'5000 afin de tester DVMS ; notre participation au challenge Grid'5000 représente à mes yeux l'accomplissement de cette collaboration, d'autant plus qu'Adrien nous a avoué avoir été sidéré par les résultats ;
– Mohammad Atiqul HAQUE, pour avoir défriché le terrain de la validation expérimentale de DVMS sur Grid'5000 en sélectionnant une pile logicielle appropriée.

Merci aux personnes qui m'ont apporté leur aide, d'un point de vue conceptuel, technique ou administratif, notamment :
– Ronan-Alexandre CHERRUEAU, Rémi DOUENCE, Adrien LÈBRE, Thomas LEDOUX, Florent MARCHAND DE KERCHOVE, Julien MERCADAL, Jonathan PASTOR, et Mario SÜDHOLT, pour vos nombreuses remarques qui m'ont permis d'améliorer substantiellement ma présentation pour la soutenance ;
– Catherine FOURNY, Florence ROGUES, Cécile DEROUET, Diana GAUDIN, Nadine PELLERAY et Hanane MAAROUFI, pour vous être montrées disponibles et avoir réagi rapidement lorsque je vous sollicitais (en particulier pour mes demandes de mission et pour l'envoi du manuscrit aux membres de mon jury) ;
– le personnel du Centre d'Appui aux Pratiques d'Enseignement, et plus particulièrement Christian COLIN, pour vos conseils qui ont permis d'organiser une webconférence dans les meilleures conditions avec les membres du jury qui ne pouvaient se déplacer ;
– Michelle DAUVÉ, pour votre diligence dans toutes les procédures impliquant l'école doctorale ;
– Thierry BARTOUX, pour avoir imprimé promptement plusieurs exemplaires de ce manuscrit afin de les envoyer aux membres du jury ;
– Fabien HERMENIER, Guillaume LE LOUËT, Frédéric DUMONT, Alexandre GARNIER et Thierry BERNARD, pour votre aide sur Entropy ;
– Arnaud LEGRAND, pour notre discussion très constructive concernant l'établissement des bases conceptuelles de DVMS ;
– Gianluigi ZANETTI, pour avoir suggéré un certain nombre de perspectives d'amélioration intéressantes pour DVMS ;
– Gilles CHABERT, pour notre coopération visant à déterminer le nombre limite d'événements que DVMS était capable de gérer simultanément ;
– Christine LOUBERRY, pour nos échanges autour des systèmes de traitement des événements.

Merci également à ceux qui m'ont permis de mûrir ma décision d'effectuer un doctorat : Hervé GRALL, Mario SÜDHOLT, Jacques NOYÉ, Jean-Marc MENAUD, Adrien LÈBRE, Marc LÉGER, Rémy POTTIER et Angel NÚÑEZ (par ordre chronologique).

Merci à toutes les personnes que j'ai croisées au volley et avec les-

quelles j'ai passé des moments très agréables, notamment : Guillaume DOUX, Alexis DE CLERCQ, Frederico ÁLVARES, Gustavo BRAND, Jonathan PASTOR, Diego TORRES, Fabian BÜTTNER, Carlos MONTOYA, Cauê CLASEN, Salvador MARTÍNEZ, Valerio COSENTINO, Marcus BLUHM, Guilhem JABERT, Swensy JANGAL, Ludivine VENDÉ, Luisa GONZALEZ, Víctor GARCÍA, Sagar SEN et Guillaume LE LOUËT. Mention spéciale à Massimo TISI, c'est grâce à toi que le groupe a évolué et a pu garder sa cohésion sur le long terme.

Merci aux personnes du département informatique que je n'ai pas encore mentionnées.

Merci Olivier NATIVEL, pour être venu à ma soutenance, même si le moment n'était pas idéal pour toi.

Merci Rémi SIFFERT, pour t'être intéressé à mes travaux et avoir fait le déplacement depuis Lyon pour assister à ma soutenance.

Enfin, merci à mes parents, pour votre écoute, votre soutien et votre présence (tout au long de la thèse et plus particulièrement avant et pendant la soutenance), ainsi que pour tous les bons moments que nous passons ensemble depuis bientôt trois décennies.

Table des matières

Liste des tableaux

Table des figures

Liste des abréviations

ACO *Ant Colony Optimization*.
API *Application Programming Interface*.

BOINC *Berkeley Open Infrastructure for Network Computing*.
BVT *Borrowed Virtual Time scheduler*.

CFS *Completely Fair Scheduler*.
CS *Credit Scheduler*.

DVMS *Distributed Virtual Machine Scheduler*.

E/S Entrées/Sorties.
EC2 *Elastic Compute Cloud*.
EGEE *Enabling Grids for E-sciencE*.
EGI *European Grid Infrastructure*.

IaaS *Infrastructure as a Service*.
IP *Internet Protocol*.

JRE *Java Runtime Environment*.
JVM *Java Virtual Machine*.

KSM *Kernel Shared Memory*.
KVM *Kernel-based Virtual Machine*.

LGIV Logiciel de Gestion d'Infrastructure Virtualisée.
LHC *Large Hadron Collider*.

MHz Mégahertz.
MPI *Message Passing Interface*.

NFS *Network File System*.
NTP *Network Time Protocol*.

OSG *Open Science Grid*.

PaaS *Platform as a Service*.

SaaS *Software as a Service*.
SCVMM *System Center Virtual Machine Manager*.
SED Système d'Exploitation Distribué.
SEG Système d'Exploitation Généraliste.

URL *Uniform Resource Locator*.

VLAN *Virtual Local Area Network*.
VM *Virtual Machine*.

WLCG *Worldwide LHC Computing Grid*.

XSEDE *Extreme Science and Engineering Discovery Environment*.

Introduction

Contexte

Les besoins croissants en puissance de calcul sont satisfaits de nos jours en fédérant de plus en plus d'ordinateurs (ou nœuds) pour former des infrastructures distribuées.

Historiquement, ces infrastructures ont été gérées d'un point de vue logiciel par des intergiciels [Fos06, LFF+06] ou des systèmes d'exploitation distribués [MvRT+90, PPD+95, LGV+05, Ril06, CFJ+08].

Ces dernières années ont vu l'apparition d'une nouvelle catégorie de gestionnaires logiciels qui reposent sur les technologies de virtualisation système [NWG+09, SMLF09, VMw10, VMw11, Apa12, Cit12, Mic12, Ope12, nim13]. La virtualisation système permet de découpler un logiciel du nœud sous-jacent en l'encapsulant dans une machine virtuelle [PG74, SN05]. Cette technologie offre des avantages conséquents à la fois aux fournisseurs et aux utilisateurs des infrastructures distribuées. Elle a ainsi favorisé l'émergence des centrales numériques qui fournissent de l'informatique en nuage (ou *Cloud Computing*) de type *Infrastructure as a Service* ; autrement dit, ces centrales mettent des machines virtuelles nues à disposition des utilisateurs, qui les personnalisent en installant un système d'exploitation et des applications.

Problématique et contributions

Les logiciels de gestion d'infrastructure virtualisée (LGIV) permettent principalement de créer de nouvelles machines virtuelles sur demande des utilisateurs, de les déployer sur des nœuds et de les gérer tout au long de leur cycle de vie.

Les LGIV sont pour la plupart hautement centralisés, c'est-à-dire que les tâches de gestion sont généralement effectuées par un nombre restreint de nœuds dédiés. Si une telle approche facilite certaines tâches d'administration et s'avère parfois nécessaire (par exemple pour avoir une vision globale du taux d'utilisation de l'infrastructure), elle peut également constituer un handicap. En effet, la centralisation limite la capacité des LGIV à passer à l'échelle, c'est-à-dire à gérer de manière réactive des infrastructures virtualisées de grande taille (plusieurs dizaines de milliers de nœuds), qui sont de plus en plus courantes de nos jours [who13].

Au cours de cette thèse, nous nous sommes intéressés aux façons d'améliorer le passage à l'échelle des LGIV ; l'une d'entre elles consiste à décentraliser le traitement de certaines tâches de gestion.

Cette problématique de décentralisation a déjà été abordée par les recherches sur les systèmes d'exploitation distribués (SED). Nous nous sommes alors demandé si ces recherches pouvaient bénéficier aux LGIV. Afin de répondre à cette question, nous avons comparé les fonctionnalités de gestion proposées par les LGIV et les SED, au niveau local et à l'échelle de l'infrastructure tout entière [QL11]. Nous avons dans un premier temps développé des réflexions initiées il y a quelques années [HWF+05, HUL06, REH07] afin de montrer que les recherches sur les systèmes d'exploitation avaient profité aux technologies de virtualisation, et inversement. Nous avons ensuite étendu notre étude au contexte distribué.

La comparaison des LGIV et des SED nous a permis d'identifier des axes de contribution, notamment en matière de décentralisation de l'ordonnancement dynamique des machines virtuelles. L'ordonnancement dynamique des machines virtuelles vise à déplacer des machines virtuelles d'un nœud à un autre lorsque cela est nécessaire, par exemple (i) pour permettre à un administrateur de réaliser une opération de maintenance ou (ii) pour optimiser l'utilisation de l'infrastructure en tenant compte de l'évolution des besoins en ressources des machines virtuelles. L'ordonnancement dynamique est encore peu utilisé dans les LGIV déployés en production, bien que plusieurs approches aient été proposées dans la littérature scientifique. Toutefois, étant donné qu'elles reposent sur un modèle centralisé, ces approches se confrontent au problème du passage à l'échelle et ne sont pas aptes à réagir rapidement lorsque certains nœuds s'avèrent surutilisés. Cela peut conduire à une violation des niveaux de qualité de service proposés aux utilisateurs, puisque les besoins en ressources des machines virtuelles ne sont pas satisfaits pendant un certain temps.

Pour pallier ce problème, plusieurs propositions ont été effectuées afin de décentraliser l'ordonnancement dynamique des machines virtuelles [BDNDM10, YMF+10, MBP11, MMP11, RC11, FME12, FRM12]. Cependant, la quasi-totalité des prototypes mis en œuvre continuent d'utiliser des mécanismes partiellement centralisés et ne répondent que de manière limitée aux besoins de réactivité et de passage à l'échelle.

C'est autour de cette problématique que s'articule la contribution majeure de cette thèse ; plus précisément, nous proposons DVMS (*Distributed Virtual Machine Scheduler*), une solution plus décentralisée pour ordonnancer dynamiquement les machines virtuelles hébergées sur une infrastructure distribuée. DVMS est déployé sous la forme d'un réseau d'agents organisés en anneau et qui coopèrent entre eux afin de traiter au plus vite des événements (liés à la surutilisation ou à la sous-utilisation des ressources d'un nœud) se produisant sur l'infrastructure ; DVMS peut traiter plusieurs événements de manière simultanée et indépendante en partitionnant dynamiquement le système, chaque partition ayant une taille appropriée à la complexité de l'événement à traiter. Nous avons optimisé le parcours de l'anneau en définissant des raccourcis permettant de transférer un message à l'extérieur d'une partition sans le faire transiter par chaque nœud de ladite partition. Par ailleurs, nous garantissons que chaque événement sera résolu si une solution existe. Pour cela, nous nous autorisons à fusionner des partitions deux à deux lorsqu'il n'y a plus de nœuds disponibles pour intégrer une partition qui a besoin de s'agrandir pour résoudre son événement ; pour qu'une fusion se déroule correctement, il est nécessaire de parvenir à un consensus entre les partitions afin d'éviter les situations d'interblocages.

Nous avons mis en œuvre ces concepts dans un prototype, que nous avons validé (i) au travers de simulations (dans un premier temps via un outil de test conçu spécifiquement pour répondre à nos besoins, puis par la suite via la boîte à outils SimGrid [CLQ08]), et (ii) par le biais de mises en situation réelle sur le banc de test Grid'5000 [gri13] (en nous appuyant sur l'outil Flauncher [BCAC+12] pour configurer les nœuds et les machines virtuelles). Nous avons pu constater grâce à ces expériences que DVMS se montrait particulièrement réactif pour gérer des infrastructures virtualisées comprenant plusieurs dizaines de milliers de machines virtuelles réparties sur des milliers de nœuds ; en effet, le temps nécessaire pour trouver une solution à un problème lié à la surutilisation des ressources d'un nœud

était de l'ordre de la seconde, là où les précédentes solutions pouvaient nécessiter jusqu'à plusieurs minutes.

Une fois le prototype validé [QL12, QLS12], nous nous sommes penchés sur les perspectives d'extension pouvant être menées autour de DVMS, notamment :

- l'enrichissement du modèle événementiel, via la définition de nouveaux événements liés par exemple à l'arrivée de nouvelles machines virtuelles sur l'infrastructure ou à la mise en maintenance d'un nœud ;
- l'ajout de mécanismes de tolérance aux pannes, afin que l'ordonnancement puisse continuer même en cas de défaillance d'un nœud ;
- la prise en compte de la topologie réseau dans la constitution des partitions, afin de faire communiquer efficacement des infrastructures distribuées reliées entre elles par un réseau étendu.

L'objectif est de disposer à terme d'un LGIV décentralisé complet, objectif qui devrait être atteint au travers de l'initiative Discovery [LAGm12] qui fait suite à ces travaux de doctorat.

Organisation du document

Le contenu de ce manuscrit est divisé en trois parties.

Première partie : gestion des infrastructures distribuées

La première partie traite de la gestion des infrastructures distribuées.

Dans le chapitre 1, nous présentons les principaux types d'infrastructures distribuées existant de nos jours, ainsi que les solutions logicielles traditionnellement utilisées pour les gérer.

Dans le chapitre 2, nous introduisons la virtualisation et expliquons les avantages qu'elle présente dans le cadre de la gestion et de l'utilisation des infrastructures distribuées.

Dans le chapitre 3, nous nous intéressons aux fonctionnalités et aux limitations des principaux logiciels de gestion d'infrastructure virtualisée.

Deuxième partie : vers une solution logicielle coopérative et décentralisée pour la gestion des infrastructures virtualisées

La deuxième partie est liée à l'étude des composants qui sont nécessaires à l'élaboration d'une solution logicielle coopérative et décentralisée pour la gestion des infrastructures virtualisées.

Dans le chapitre 4, nous étudions les similitudes entre les logiciels de gestion d'infrastructure virtualisée et les solutions traditionnelles de gestion d'infrastructure distribuée, et nous identifions des axes de contribution, principalement autour de l'ordonnancement des machines virtuelles.

Dans le chapitre 5, nous recentrons l'étude précédente sur les dernières contributions en matière d'ordonnancement dynamique décentralisé de machines virtuelles.

Troisième partie : DVMS, une solution coopérative et décentralisée pour l'ordonnancement dynamique des machines virtuelles

Dans la troisième partie, nous traitons de la contribution majeure de cette thèse : DVMS (*Distributed Virtual Machine Scheduler*), une solution logicielle coopérative et décentralisée pour l'ordonnancement dynamique des machines virtuelles.

Dans le chapitre 6, nous présentons les fondements théoriques de l'approche et la mise en œuvre du prototype correspondant.

Dans le chapitre 7, nous détaillons le protocole expérimental et les outils mis en œuvre afin d'évaluer et de valider le bon fonctionnement du prototype DVMS.

Dans le chapitre 8, nous analysons les résultats des expériences menées avec DVMS.

Dans le chapitre 9, nous explorons plusieurs perspectives de travaux autour de DVMS.

Diffusion scientifique

Les travaux détaillés dans ce document ont fait l'objet de plusieurs présentations, dans le cadre de séminaires internes et externes, notamment au laboratoire informatique de Grenoble (décembre 2010), au cours d'une journée d'échange entre doctorants (avril 2011), et devant l'équipe Myriads d'INRIA Rennes (mars 2012).

Par ailleurs, le bon fonctionnement de DVMS a été démontré en direct au travers d'une mise en situation réelle sur le banc de test Grid'5000 lors de l'école d'hiver éponyme (décembre 2012) ; à cette occasion, 960 machines virtuelles ont été créées et ordonnancées avec succès sur 132 nœuds en l'espace de 20 minutes.

Enfin, ces travaux ont donné lieu à plusieurs publications qui sont mentionnées ci-dessous.

Revue scientifique avec comité de lecture

Flavien Quesnel, Adrien Lèbre, and Mario Südholt. Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS. *Concurrency and Computation: Practice and Experience*, 2012.

Conférence internationale avec comité de lecture et publication des actes

Flavien Quesnel and Adrien Lèbre. Operating Systems and Virtualization Frameworks: From Local to Distributed Similarities. In *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.

Ateliers internationaux avec comité de lecture et publication des actes

Adrien Lèbre, Paolo Anedda, Massimo Gaggero, and Flavien Quesnel. Discovery, beyond the clouds. In *Euro-Par 2011: Parallel Processing Workshops*, volume 7156 of *Lecture Notes in Computer Science*, pages 446–456. Springer, Berlin/Heidelberg, Germany, 2012.

Flavien Quesnel and Adrien Lèbre. Cooperative Dynamic Scheduling of Virtual Machines in Distributed Systems. In *Euro-Par 2011: Parallel Processing Workshops*, volume 7156 of *Lecture Notes in Computer Science*, pages 457–466. Springer, Berlin/Heidelberg, Germany, 2012.

Rapport technique

Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. Adding Virtualization Capabilities to Grid'5000. Technical Report RR-8026, INRIA, July 2012.

Gestion des infrastructures distribuées

1

Les infrastructures distribuées avant l'essor de la virtualisation

Les organismes ayant besoin d'une grande puissance de calcul ont la possibilité de recourir soit à des ordinateurs centraux très performants (les *mainframes*), soit à des fédérations d'ordinateurs moins puissants (appelés nœuds), qui forment des infrastructures distribuées. Cette dernière solution est devenue prépondérante ces dernières années, ce qui peut s'expliquer par des raisons économiques (à puissance égale, une fédération de nœuds est moins coûteuse qu'un ordinateur central) et par le fait qu'une fédération comportant un grand nombre de nœuds offre plus de puissance de calcul qu'un ordinateur central.

Dans ce chapitre, nous présentons les principales catégories d'infrastructures distribuées, et nous nous intéressons à leur gestion d'un point de vue logiciel. Nous abordons en particulier les logiciels de gestion qui ont été conçus pour ces infrastructures avant l'essor de la virtualisation, sur laquelle nous reviendrons au chapitre suivant.

Sommaire

1.1 Présentation des infrastructures distribuées

Les premières infrastructures distribuées à être apparues sont les grappes de nœuds ; elles ont été suivies par les centres de données et de calculs, les grilles et les plates-formes de calcul partagé reposant sur le volontariat (cf. figure 1.1).

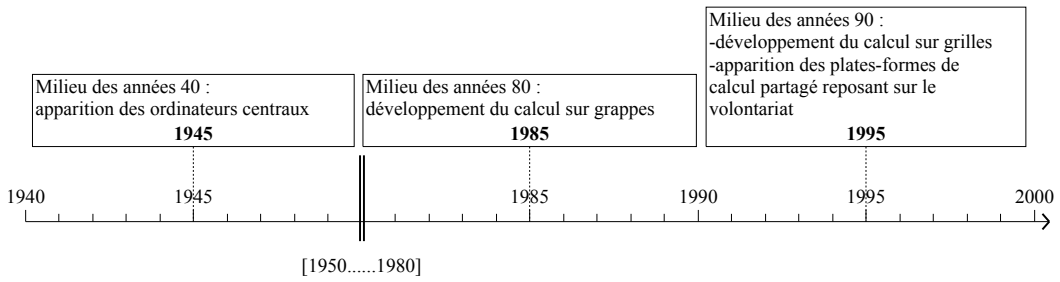


FIGURE 1.1 – Chronologie d'apparition des principales catégories d'infrastructures distribuées

1.1.1 Grappe de nœuds

L'unité de base généralement utilisée dans les infrastructures distribuées est la grappe (ou ferme) de nœuds.

Définition 1.1 (Grappe)
Une grappe (ou *cluster*) est une fédération de nœuds homogènes, c'est-à-dire que les nœuds sont tous identiques, afin de faciliter aussi bien leur maintenance que leur utilisation. Ces nœuds sont géographiquement très proches (typiquement la même pièce) et sont reliés entre eux par un réseau local haute performance.

1.1.2 Centre de données et de calculs

Les grappes ne sont pas forcément utilisées de manière indépendante, mais peuvent au contraire être regroupées au sein d'une fédération, par exemple un centre de données et de calculs.

Définition 1.2 (Centre de données et de calculs)
Un centre de données et de calculs (ou *datacenter*) est une fédération de grappes particulière dans laquelle les grappes sont géographiquement proches les unes des autres (typiquement le même bâtiment ou le même groupe de bâtiments) et communiquent entre elles au travers d'un réseau local.

Le type de nœuds peut varier d'une grappe à l'autre, notamment si les grappes n'ont pas été mises en service en même temps.

Chaque grappe dispose de son propre réseau, et tous les réseaux n'ont pas nécessairement la même performance.

1.1.3 Grille

Les grappes et les centres de données de plusieurs organismes partageant un même but peuvent être mis en commun afin de constituer une infrastructure disposant d'une puissance de calcul accrue, appelée grille.

Définition 1.3 (Grille)
Une grille est une infrastructure distribuée qui « permet le partage de ressources et la résolution coordonnée de problèmes au sein d'organisations virtuelles multi-institutionnelles » [FZRL08].

Une grille est généralement une fédération de nœuds hétérogènes.

Par ailleurs, les différentes composantes d'une grille communiquent via un réseau étendu, qui est moins performant qu'un réseau local ; ceci est vrai pour la latence (c'est-à-dire le temps nécessaire pour transmettre un message entre deux nœuds distants), et parfois également pour le débit (c'est-à-dire la quantité de données que deux nœuds distants peuvent s'échanger par unité de temps).

Il existe de nombreux exemples de grilles. Certaines ont été mises en place à l'échelle nationale, comme Grid'5000 [gri13] (cf. figure 1.2(a)) et l'infrastructure gérée par France Grilles [fra13] en France, ou FutureGrid [fut13], *Open Science Grid* (OSG) [osg13] et *Extreme Science and Engineering Discovery Environment* (XSEDE, anciennement TeraGrid) [xse13] aux États-Unis. D'autres se sont développées sur tout un continent, en s'appuyant sur les grilles nationales, comme c'est le cas pour la *European Grid Infrastructure* (EGI, anciennement EGEE – *Enabling Grids for E-sciencE* –) [egi13] en Europe. D'autres enfin sont d'échelle mondiale, comme la *Worldwide LHC Computing Grid* (WLCG) [wlc13] (cf. figure 1.2(b)), qui repose notamment sur les grilles OSG et EGI pour analyser les données issues du LHC (*Large Hadron Collider*), l'accélérateur de particules du CERN (Centre Européen pour la Recherche Nucléaire).

(a) Grid'5000

(b) WLCG

FIGURE 1.2 – Répartition géographique des infrastructures composant une grille

1.1.4 Plate-forme de calcul partagé reposant sur le volontariat

Lorsque les ressources de calcul sont mises en commun par des individus plutôt que par des institutions, cela donne un nouveau type d'infrastructure distribuée que nous désignerons sous l'expression de plate-forme de calcul partagé reposant sur le volontariat.

Définition 1.4 (Plate-forme de calcul partagé reposant sur le volontariat)
Une plate-forme de calcul partagé reposant sur le volontariat peut être vue comme une variante d'une grille, à ceci près qu'elle est composée de nœuds hétérogènes mis à disposition par des volontaires (pas nécessairement des institutions) et qui sont typiquement reliés entre eux via le réseau Internet.

BOINC (*Berkeley Open Infrastructure for Network Computing*) [And04] est un exemple d'une telle plate-forme. Elle vise à regrouper des internautes autour de différents projets de recherche, dont le plus célèbre est probablement SETI@home [set13]. SETI@home a pour objectif d'analyser les communications radio en provenance de l'espace à la recherche de signes d'intelligence extra-terrestre. Les internautes doivent simplement télécharger le logiciel BOINC et s'inscrire au projet auquel ils souhaitent participer ; lorsqu'ils n'utilisent pas leur ordinateur, le logiciel va récupérer automatiquement des tâches à accomplir (par exemple des calculs à effectuer ou des données à analyser) auprès du projet en question, les traiter, puis soumettre au projet les résultats obtenus.

XtremWeb [FGNC01] est un logiciel permettant de mettre en place une plate-forme similaire à BOINC. Cependant, contrairement à BOINC, il permet à des tâches qui sont réparties sur les ordinateurs de plusieurs utilisateurs de communiquer directement entre elles.

1.2 Gestion logicielle des infrastructures distribuées

La gestion des infrastructures distribuées introduites précédemment nécessite de prendre en compte un certain nombre de problématiques, notamment la connexion des utilisateurs au système et leur identification, la soumission de tâches, leur ordonnancement, leur déploiement, la surveillance de leur bon fonctionnement, ainsi que leur terminaison.

Ces différentes problématiques peuvent impliquer plusieurs catégories de ressources (cf. figure 1.3) :

- les nœuds d'accès, pour la connexion des utilisateurs ;
- un ou plusieurs nœuds dédiés à la gestion de l'infrastructure ;
- un espace de stockage, pour les données des utilisateurs ;
- les nœuds de calcul, pour traiter les tâches soumises par les utilisateurs.

1.2.1 Connexion sécurisée à l'infrastructure et identification des utilisateurs

Afin de pouvoir utiliser une infrastructure, les utilisateurs doivent tout d'abord se connecter à celle-ci [LFF+06, CFJ+08, gri13].

Cette connexion peut être réalisée de plusieurs façons. D'un point de vue matériel, l'utilisateur est susceptible de se connecter au travers d'un réseau privé ou via

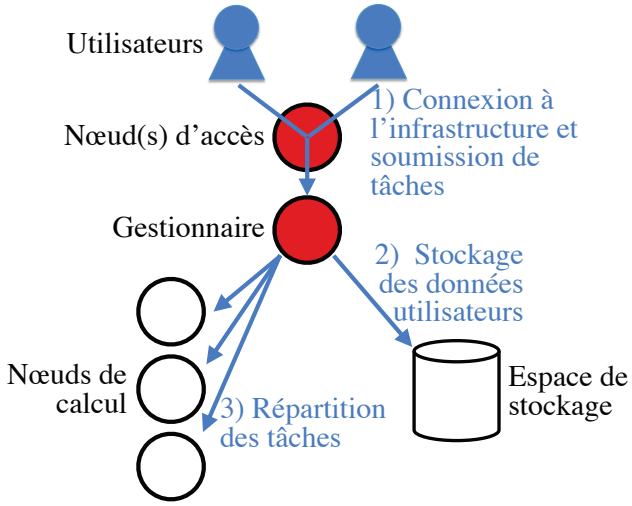


FIGURE 1.3 – Exemple d'organisation d'une infrastructure distribuée

le réseau Internet ; par ailleurs il peut être autorisé à se connecter à tous les nœuds de l'infrastructure, ou seulement à des nœuds dédiés à cette tâche (les nœuds d'accès). D'un point de vue logiciel, il faut déterminer quel logiciel et quel protocole de communication utiliser pour mettre en relation l'utilisateur et le système ; ce choix est critique pour la sécurité de l'infrastructure, afin d'identifier les utilisateurs, de savoir à quelles ressources ils peuvent accéder et pour combien de temps, de comptabiliser les ressources qu'ils ont déjà consommées par le passé, et d'empêcher qu'un utilisateur malicieux dérobe les ressources ou les données d'un autre utilisateur.

1.2.2 Soumission de tâches

Une fois connecté à l'infrastructure, un utilisateur doit pouvoir lui soumettre des tâches à réaliser [LFF+06, CFJ+08, gri13].

Pour cela, il doit spécifier les caractéristiques des tâches, en fonction des possibilités offertes par l'infrastructure :

- les programmes et/ou données nécessaires pour mener les tâches à bien ; au besoin, l'utilisateur doit pouvoir les déposer sur l'infrastructure ;
- les ressources nécessaires, d'un point de vue qualitatif et quantitatif, et la durée pendant laquelle elles seront utilisées ; il faut également indiquer si les tâches peuvent être éventuellement accomplies dans un mode dégradé, c'est-à-dire avec moins de ressources que ce qui est demandé ;
- l'heure à laquelle le traitement de la tâche doit commencer, et celle à laquelle il doit être terminé ;
- les liens entre les différentes tâches (s'il y en a), et les éventuelles contraintes d'antériorité, qui spécifient que certaines tâches doivent être accomplies avant que le traitement d'autres tâches puisse débuter ; dans le cas où les tâches sont liées entre elles, le logiciel de gestion de l'infrastructure doit prendre garde à exécuter des actions cohérentes sur l'ensemble des tâches, et ce de manière simultanée ; le terme d'ordonnancement est alors utilisé.

1.2.3 Ordonnancement des tâches

Définition 1.5 (Ordonnancement)
L'ordonnancement est l'action d'assigner des ressources au traitement des tâches [Rot94, Tan01, Leu04, Sta08]. L'ordonnancement est réalisé par une entité logicielle appelée ordonnanceur.

L'ordonnancement doit tenir compte des caractéristiques des tâches mentionnées précédemment : ressources nécessaires, heure de début et de fin, priorité, lien entre les tâches, etc.

L'ordonnancement peut être statique ou dynamique.

Définition 1.6 (Ordonnancement statique)
L'ordonnancement est qualifié de statique lorsque chaque tâche demeure sur le même nœud de calcul du début à la fin de son traitement. Le placement initial des tâches tient compte de la demande en ressources formulée par les utilisateurs, et non pas des besoins réels en ressources de ces tâches.

Définition 1.7 (Ordonnancement dynamique)
L'ordonnancement est qualifié de dynamique lorsque les tâches peuvent être migrées d'un nœud de calcul à un autre durant leur traitement, ce qui se fait en tenant compte de leurs besoins réels en ressources.

L'ordonnancement vise à remplir un ou plusieurs objectifs. Certains objectifs touchent à la rapidité de traitement des tâches. D'autres sont liés à la répartition équitable des ressources entre les tâches. D'autres encore sont relatifs à l'utilisation optimale des ressources, par exemple l'équilibrage de la charge de travail entre les ressources, ou au contraire sa consolidation sur un nombre restreint de ressources afin de maximiser l'utilisation de ces dernières. D'autres encore visent à respecter des contraintes de placement, qui peuvent par exemple découler des affinités ou des antagonismes entre les tâches. Enfin, certains objectifs peuvent consister à respecter d'autres types de contraintes, comme les contraintes d'antériorité.

L'ordonnancement peut tenir compte de la volatilité de l'infrastructure, autrement dit de l'ajout ou du retrait de ressources. Cet ajout ou retrait peut être volontaire, si les propriétaires de l'infrastructure souhaitent mettre plus de ressources à disposition de leurs utilisateurs, retirer des ressources obsolètes, ou encore soustraire temporairement des ressources pour réaliser une opération de maintenance (par exemple pour mettre à jour des logiciels, ou remplacer des composants défectueux). Le retrait de ressources peut également s'avérer accidentel en cas de défaillance matérielle ou logicielle, ce qui a d'autant plus de risques de se produire que l'infrastructure est grande.

1.2.4 Déploiement des tâches

Une fois que l'ordonnanceur a décidé quelles ressources attribuer à une tâche, il faut encore déployer cette dernière sur le nœud de calcul approprié.

Cela peut nécessiter l'installation et la configuration d'un environnement logiciel adapté, en plus de la copie des programmes et des données nécessaires au traitement de la tâche.

Les données associées à la tâche peuvent être stockées en local sur le nœud de calcul ou bien à distance, sur un serveur de stockage partagé, sur un ensemble de

nœuds hébergeant un système de fichiers répartis, ou bien dans une baie de stockage (c'est-à-dire une armoire ne contenant que des disques durs).

1.2.5 Surveillance de l'état de l'infrastructure

Chaque tâche est susceptible d'être migrée d'un nœud de calcul à un autre si l'ordonnancement est dynamique ; pour cela, l'ordonnanceur s'appuie sur la collecte d'informations sur l'état des ressources (ou *monitoring*) ; cette collecte présente également un intérêt en dehors du contexte de l'ordonnancement.

Elle permet tout d'abord aux administrateurs d'avoir des informations sur l'état de l'infrastructure, et notamment d'être prévenu en cas de défaillances matérielles, voire logicielles.

Elle sert également à comptabiliser la consommation en ressources de chaque utilisateur, afin de vérifier que chacun respecte les conditions d'utilisation de l'infrastructure.

Enfin, elle autorise les utilisateurs à connaître l'état d'avancement de leurs tâches : en attente de traitement, en cours de traitement, ou accomplie.

1.2.6 Terminaison des tâches

Une fois que ses tâches sont accomplies, un utilisateur doit pouvoir récupérer les résultats issus de leur traitement.

Par la suite, il peut être nécessaire de procéder à un « nettoyage » des ressources pour revenir à une configuration par défaut, afin qu'elles puissent être utilisées sans problème pour traiter d'autres tâches.

1.3 Solutions logicielles traditionnellement utilisées pour la gestion des infrastructures distribuées

Les concepts présentés précédemment sont mis en œuvre par les solutions logicielles traditionnellement utilisées pour gérer les infrastructures distribuées, à savoir les intergiciels [Fos06, LFF+06] et les systèmes d'exploitation distribués [MvRT+90, PPD+95, LGV+05, Ril06, CFJ+08].

1.3.1 Intergiciels

Les intergiciels constituent une solution très populaire pour gérer les infrastructures distribuées.

Définition 1.8 (Intergiciel)
Un intergiciel est un logiciel construit au-dessus d'un système d'exploitation existant.

Les intergiciels les plus complets, tels que Globus [Fos06] ou gLite [LFF+06], permettent de gérer des grilles ; d'ailleurs, gLite a été conçu à l'origine pour tirer parti de l'ex-grille européenne EGEE, qui a été mentionnée précédemment.

Ces intergiciels reposent sur l'utilisation d'ordonnanceurs par lots (ou *batch schedulers*), ceux-ci pouvant être indissociables des intergiciels ou bien constituer des projets indépendants, comme c'est le cas par exemple pour Condor [TTL05], Torque/PBS [Ada12] ou OAR [CDCG+05]. Un ordonnanceur par lots cherche à

maximiser le taux d'utilisation des ressources de l'infrastructure. Généralement, il fait de l'ordonnancement statique et suit une approche centralisée ou hiérarchique (l'ordonnanceur réside respectivement sur un nœud, ou sur un nombre restreint de nœuds organisés de manière hiérarchique).

1.3.2 Systèmes d'exploitation distribués

Les systèmes d'exploitation distribués constituent une autre solution pour gérer les infrastructures distribuées.

Définition 1.9 (Système d'exploitation distribué)
Un système d'exploitation distribué (SED) est conçu dans l'optique d'intégrer les fonctionnalités de gestion des infrastructures distribuées au sein du système d'exploitation, dans un souci de performance et de simplicité d'utilisation [CFJ+08]. Il peut avoir été conçu à partir de zéro, ou bien à partir d'un système d'exploitation existant (et non distribué) qui a été modifié en profondeur.

Certains SED sont plus particulièrement conçus avec l'objectif de créer des systèmes à image unique.

Définition 1.10 (Système à image unique)
Un système à image unique est un système « qui masque la nature hétérogène et distribuée des ressources disponibles et qui les présente aux utilisateurs et aux applications sous la forme d'une ressource de calcul unique » [BCJ01].

Il existe un grand nombre de SED, dont Amoeba [MvRT+90], Plan 9 [PPD+95], OpenMosix [LGV+05], OpenSSI [LGV+05], Kerrighed [LGV+05], Vigne [Ril06] et XtreemOS [CFJ+08]. Certains SED sont dédiés aux grilles (tels que Vigne et XtreemOS) et d'autres aux grappes (comme Amoeba, Plan 9, Mosix, OpenSSI et Kerrighed). Notons au passage qu'un SED de grille peut s'appuyer sur un SED de grappe, comme c'est le cas pour XtreemOS avec Kerrighed.

Plusieurs SED de grappe (en particulier Mosix, OpenSSI et Kerrighed) réalisent de l'ordonnancement dynamique de tâches, et ce de manière plus décentralisée que les ordonnanceurs par lots, étant donné que le travail d'ordonnancement est réparti entre tous les nœuds de calcul. Dans le cas de Mosix, d'OpenSSI et de Kerrighed, le critère d'ordonnancement par défaut est l'équilibrage de la charge de travail des processeurs. Malheureusement, ces SED ne sont pas capables de migrer/déplacer certains types de tâches, notamment celles qui sont fortement dépendantes des ressources du nœud de calcul sur lequel elles ont été placées initialement (par exemple si elles ont besoin d'accéder directement à la carte graphique ou à une carte réseau).

Le développement des SED a été peu à peu abandonné, notamment du fait de leur complexité, qui les rend difficiles à maintenir et à mettre à jour. Ceci a profité non seulement aux intergiciels introduits précédemment, mais également aux intergiciels de nouvelle génération qui s'intéressent à la gestion des infrastructures virtualisées.

1.4 Conclusion

Dans ce chapitre, nous avons présenté les principales catégories d'infrastructures distribuées existant de nos jours, à savoir : les grappes de nœuds, les centres

de données et de calculs, les grilles et les plates-formes de calcul partagé reposant sur le volontariat.

Nous avons ensuite identifié les fonctionnalités proposées par la plupart des logiciels en charge de la gestion d'une infrastructure distribuée : la connexion sécurisée des utilisateurs à l'infrastructure, la soumission de tâches, l'ordonnancement de ces tâches, leur déploiement sur les ressources qui leur sont attribuées, la surveillance de leur état d'avancement et leur terminaison.

Nous avons enfin décrit les solutions logicielles traditionnellement utilisées pour gérer ces infrastructures ; ces solutions se présentent sous la forme d'intergiciels ou de systèmes d'exploitation distribués.

Dans le chapitre suivant, nous verrons en quoi la virtualisation a révolutionné la gestion et l'usage des infrastructures distribuées, pour donner naissance à un nouveau paradigme de calcul : l'informatique en nuage.

2

Apports de la virtualisation

La virtualisation [SN05] est une technique qui permet (i) de découpler les couches logicielles supérieures des couches logicielles inférieures et/ou du matériel, et (ii) de leurrer les premières concernant les caractéristiques réelles des seconds.

Son utilisation s'est développée dans les années soixante [Cre81]. Les prérequis matériels nécessaires à son emploi ont ensuite été formalisés au milieu des années soixante-dix [PG74]. Ces dernières années, elle est de plus en plus utilisée dans les infrastructures distribuées en raison des avantages qu'elle apporte en matière de gestion et d'utilisation des ressources.

Dans ce chapitre, nous abordons les principaux concepts relatifs à la virtualisation et nous nous intéressons à ses apports en matière de gestion et d'utilisation des ressources dans les infrastructures distribuées, apports qui ont contribué à l'essor des centrales numériques et de l'informatique en nuage.

Sommaire

2.1 Présentation de la virtualisation

2.1.1 Virtualisation système et applicative

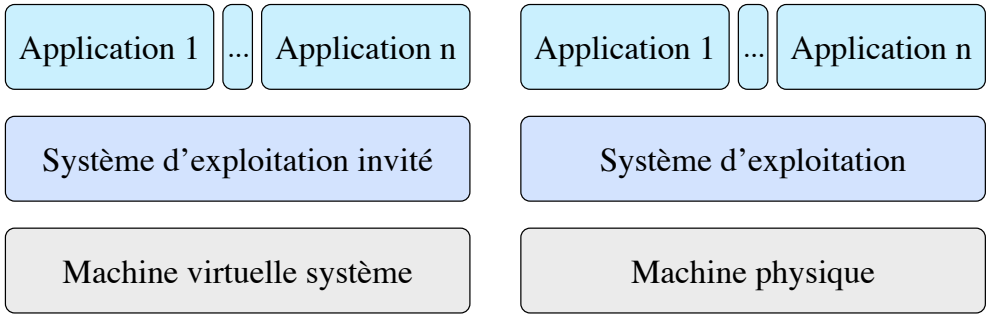
Il existe deux grands types de virtualisation [SN05] : la virtualisation système et la virtualisation applicative.

Virtualisation système

La virtualisation système vise à virtualiser uniquement le matériel.

Définition 2.1 (Machine virtuelle système, au sens strict)
Une *machine virtuelle système* est un équivalent logiciel d'une *machine physique* donnée, soit une agrégation de processeurs, de mémoire et de périphériques (disque dur, carte réseau, carte graphique, etc.).

Afin de tirer parti d'une machine virtuelle système, il est nécessaire d'installer des applications et un système d'exploitation, système d'exploitation qui est qualifié d'*invité* pour souligner le fait qu'il n'est pas installé sur une machine physique (cf. figure 2.1).



(a) Machine virtuelle système (b) Machine physique

FIGURE 2.1 – Comparaison entre une machine virtuelle système et une machine physique

Définition 2.2 (Machine virtuelle système, au sens large)
Par abus de langage, l'expression *machine virtuelle système* est généralement utilisée pour désigner non seulement la machine virtuelle à proprement parler, mais également le système d'exploitation invité et les applications qu'elle héberge.

Les machines virtuelles sont elles-mêmes hébergées par des machines physiques sur lesquelles un hyperviseur est installé.

Définition 2.3 (Hyperviseur)
Un hyperviseur est une entité logicielle chargée de l'attribution des ressources d'une ou plusieurs machines physiques aux machines virtuelles, ainsi que de la gestion de ces dernières.

Les hyperviseurs peuvent être regroupés en deux catégories : les hyperviseurs natifs (dits *de type I*) et les hyperviseurs hébergés (dits *de type II*).

Définition 2.4 (Hyperviseur natif, dit de type I)
Un hyperviseur natif (de type I) est « l'unique logiciel qui s'exécute avec le plus haut niveau de privilège autorisé par l'architecture de la machine » physique [SN05].

Définition 2.5 (Hyperviseur hébergé, dit de type II)
Un hyperviseur hébergé (de type II) est installé « sur une plate-forme hôte qui fait déjà fonctionner un système d'exploitation » et « utilise les fonctionnalités qui sont proposées par le système d'exploitation hôte pour contrôler et gérer les ressources demandées par chaque machine virtuelle » [SN05].
Un hyperviseur hébergé est susceptible de s'exécuter avec le plus haut niveau de privilège (conjointement avec le système d'exploitation hôte), ou bien de s'exécuter partiellement ou entièrement avec un niveau de privilège moins élevé.

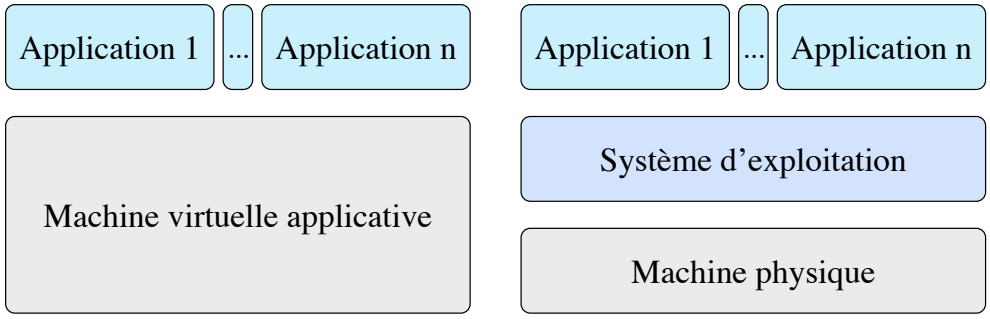
VMware ESX [Wal02] et Citrix XenServer [Cit12] sont des exemples d'hyperviseurs natifs, tandis que Red Hat KVM (*Kernel-based Virtual Machine*) [KKL+07] et Microsoft Hyper-V [CBER09] sont des hyperviseurs hébergés.

Virtualisation applicative

Contrairement à la virtualisation système, la virtualisation applicative ne se contente pas de virtualiser la machine physique.

Définition 2.6 (Machine virtuelle applicative)
Une *machine virtuelle applicative* est un équivalent logiciel d'une machine physique et de tout ou partie d'un système d'exploitation.

Pour utiliser une machine virtuelle applicative, il suffit d'installer les applications et la portion du système d'exploitation qui n'est pas virtualisée (cf. figure 2.2).



(a) Machine virtuelle applicative (b) Machine physique

FIGURE 2.2 – Comparaison entre une machine virtuelle applicative et une machine physique

De la même manière que pour les machines virtuelles système, les machines virtuelles applicatives sont hébergées sur des machines physiques. En revanche, au lieu d'être gérées par un hyperviseur, elles sont administrées par le système d'exploitation.

Les deux exemples les plus connus de mise en œuvre de la virtualisation applicative sont :

- les conteneurs [SPF+07, BBHL08], qui servent généralement à partitionner les ressources de la machine physique sous-jacente, chaque conteneur pouvant exécuter plusieurs applications ;
- les machines virtuelles dédiées à l'exécution d'applications écrites dans des langages de programmation de haut niveau, comme c'est le cas avec la *Java Virtual Machine* (JVM) [LY99] ; grâce à la virtualisation applicative, un même programme Java peut fonctionner sur tous les systèmes d'exploitation qui disposent d'une JVM ; en revanche, contrairement à un conteneur, une JVM ne permet d'exécuter qu'un seul programme Java ; pour lancer plusieurs programmes, il est nécessaire de démarrer autant de JVM.

Par la suite, nous nous intéresserons uniquement à la virtualisation système.

2.1.2 Abstractions créées par les hyperviseurs

La virtualisation système permet à un hyperviseur de leurrer les logiciels installés sur une machine virtuelle quant aux caractéristiques réelles des ressources physiques sous-jacentes. Ces ressources peuvent être abstraites de trois façons par l'hyperviseur : *traduction*, *agrégation de ressources* ou *partitionnement de ressources*.

Traduction

Si le processeur de la machine physique et celui de la machine virtuelle n'appartiennent pas à la même famille (ou architecture), l'hyperviseur doit alors procéder à une traduction des instructions exécutées par la machine virtuelle (pour le compte du système d'exploitation invité et des applications qu'elle héberge) en instructions compréhensibles par la machine physique (cf. figure 2.3(a)) [Bel05].

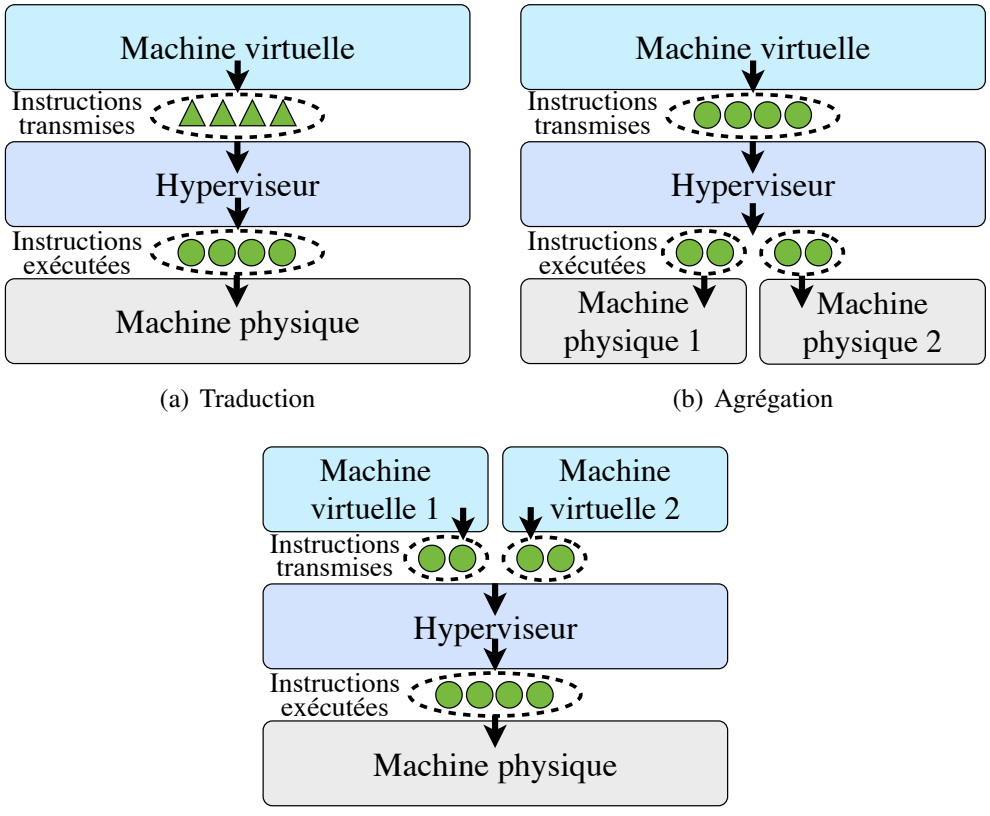
Agrégation de ressources

L'hyperviseur réalise de l'agrégation de ressources s'il permet à un système d'exploitation invité et à des applications de fonctionner sur plusieurs machines physiques, en leur donnant l'illusion d'une seule grosse machine (cf. figure 2.3(b)) [CH09].

Partitionnement de ressources

À l'inverse, l'hyperviseur accomplit du partitionnement de ressources s'il autorise plusieurs machines virtuelles à être hébergées sur une même machine physique. Le système d'exploitation invité et les applications abrités par chaque machine virtuelle ont l'illusion qu'ils sont les seuls à avoir accès aux ressources de la machine physique (cf. figure 2.3(c)) [Wal02, BDF+03, KKL+07, CBER09].

Nous ne considérerons désormais que le cas du partitionnement de ressources.



(a) Traduction (b) Agrégation

(c) Partitionnement

FIGURE 2.3 – Abstraction des ressources physiques

2.1.3 Techniques de virtualisation utilisées par les hyperviseurs

Les hyperviseurs peuvent recourir à plusieurs techniques de virtualisation pour faire fonctionner un système d'exploitation invité sur une machine virtuelle système.

Ceci peut constituer un défi suivant l'architecture du processeur de la machine physique sous-jacente, architecture qui ne présente pas nécessairement tous les prérequis identifiés par Popek et Goldberg [PG74] pour être aisément virtualisée. L'architecture x86, qui est la plus répandue dans les ordinateurs à l'heure actuelle, est ainsi difficilement virtualisable [RI00].

Pour comprendre l'origine du problème, il faut savoir qu'un système d'exploitation dispose en temps normal des pleins pouvoirs sur la machine physique sous-jacente ; autrement dit, il a le droit d'exécuter n'importe quelle instruction. Or, dans le cas de la virtualisation système, c'est à l'hyperviseur que reviennent ces pleins pouvoirs, le système d'exploitation invité n'étant gratifié que de privilèges limités. Cette transmission de préséance est effectuée par l'hyperviseur au moyen d'une ou plusieurs techniques de virtualisation, à savoir : l'émulation [Bel05], la paravirtualisation [BDF+03, Rus07] et la virtualisation assistée par le matériel (abrégée par la suite en *virtualisation matérielle*) [UNR+05, KKL+07, CBER09, LPH+10, SK10].

Émulation

L'émulation est la première technique à s'être développée pour virtualiser l'architecture x86.

Le recours à l'émulation implique que l'hyperviseur remplace dynamiquement les instructions qui proviennent du système d'exploitation invité, et qui ont besoin du plus haut niveau de privilège pour s'exécuter normalement, par des instructions équivalentes qui ne poseront aucun problème.

Paravirtualisation

Les performances des premiers hyperviseurs utilisant l'émulation n'étaient pas optimales, c'est pourquoi une nouvelle technique de virtualisation a vu le jour : la paravirtualisation.

Dans le cas de la paravirtualisation, l'hyperviseur propose une interface particulière au système d'exploitation invité pour lui permettre d'exécuter correctement des instructions privilégiées.

Pour que le système d'exploitation invité puisse tirer parti de cette interface, il doit être modifié au préalable, modification qui constitue une opération longue et complexe. Un tel système d'exploitation modifié est qualifié de *paravirtualisé*.

Virtualisation matérielle

Face à l'engouement autour de la virtualisation, les constructeurs de processeurs x86 ont décidé de concevoir des extensions matérielles destinées à faciliter le développement de nouveaux hyperviseurs.

Ces extensions se chargent d'exécuter les instructions privilégiées du système d'exploitation invité sans qu'il soit nécessaire de modifier celui-ci ou d'user de l'émulation.

Notons que les hyperviseurs tendent de plus en plus à combiner les trois techniques de virtualisation présentées dans cette section. La virtualisation matérielle est majoritairement utilisée par les hyperviseurs récents pour virtualiser le processeur et la mémoire, car c'est la technique la plus simple à mettre en œuvre [KKL+07, CBER09, LPH+10, SK10]. Concernant la virtualisation des périphériques, les hyperviseurs offrent généralement le choix entre l'émulation et la paravirtualisation, cette dernière étant à privilégier pour obtenir de meilleures performances [NAMG09].

2.1.4 Principales fonctionnalités offertes par les hyperviseurs

Indépendamment des techniques de virtualisation utilisées par les hyperviseurs, ces derniers proposent un certain nombre de fonctionnalités ; dans cette section, nous nous concentrerons sur celles qui sont les plus intéressantes pour la gestion des ressources d'une infrastructure distribuée.

Limitation des ressources utilisées

Un hyperviseur peut offrir plusieurs possibilités pour limiter la consommation en ressources des machines virtuelles, ce qui permet par exemple de garantir un partage équitable de ces ressources.

Xen permet notamment d'assigner un poids à une machine virtuelle, celle-ci étant alors autorisée à utiliser le processeur pour une durée proportionnelle à ce poids [Chi07].

Par ailleurs, tout hyperviseur de type II utilisant Linux comme système d'exploitation hôte a l'opportunité d'utiliser les cgroups [MJL13, PLS13], l'idée étant d'assigner une ou plusieurs machines virtuelles à un cgroup donné.

Concernant le processeur, il est possible d'assigner un poids à chaque cgroup ; un cgroup de poids 2 sera ainsi autorisé à accéder deux fois plus longtemps au processeur qu'un cgroup de poids 1. Par ailleurs, sur une machine physique disposant de plusieurs processeurs, les cgroups permettent de définir quels processeurs sont utilisables par les éléments d'un cgroup.

Pour ce qui est de la mémoire, il est possible de limiter la quantité utilisée ou encore les portions de mémoire accessibles par les membres d'un cgroup.

En matière de réseau, les cgroups permettent d'attribuer un poids à toute interface réseau, afin de restreindre le trafic sortant de chacune d'elle.

Enfin, dans le cas des périphériques de type blocs, il est possible d'affecter un poids à un cgroup afin d'offrir à ses membres un accès proportionnel à l'ensemble des périphériques ou bien à un périphérique donné. Par ailleurs, il est envisageable de limiter le nombre d'opérations de lecture ou d'écriture, ainsi que la quantité de données transférées en lecture ou en écriture.

Optimisation de l'utilisation de la mémoire

Les fonctionnalités présentées jusqu'à présent visaient exclusivement à restreindre l'utilisation des ressources des machines virtuelles, sans chercher à l'optimiser. L'optimisation de la consommation mémoire permet par exemple (i) d'éviter une pénurie qui pénaliserait les performances des machines virtuelles, voire (ii) de démarrer plus de machines virtuelles sur une même machine physique.

Une première optimisation consiste à faire en sorte que l'hyperviseur récupère le contrôle des portions (les *pages*) de mémoire inutilisées par les machines virtuelles ; ceci est possible grâce au *ballooning* [Wal02]. Afin de récupérer les pages mémoire inutilisées par une machine virtuelle, l'hyperviseur demande à une application spéciale installée sur celle-ci, le « ballon », de « gonfler » ; le ballon s'exécute en priant le système d'exploitation invité de la machine virtuelle de lui allouer de la mémoire ; enfin, le ballon avertit l'hyperviseur des pages mémoire qu'il détient, et que l'hyperviseur peut réutiliser à sa guise. Le processus est réversible afin de permettre à l'hyperviseur de restituer de la mémoire au système d'exploitation invité si celui-ci en a besoin.

Une autre optimisation réside dans la *déduplication mémoire* [Wal02, AEW09]. Le fonctionnement est le suivant : l'hyperviseur analyse le contenu des pages mémoire détenues par chaque machine virtuelle ; lorsqu'il découvre des pages identiques, il n'en conserve qu'une seule copie et réutilise les doublons à sa convenance.

Suspension et reprise de l'exécution

La bonne gestion des ressources utilisées par les machines virtuelles n'est pas le seul domaine d'action de l'hyperviseur ; celui-ci peut en outre autoriser la suspension de l'exécution d'une machine virtuelle, et sa reprise un peu plus tard.

L'état de la machine virtuelle est susceptible d'être conservé en mémoire, ou bien d'être stocké sur le disque dur de la machine physique hébergeant la machine virtuelle [PEL11]. Cette dernière possibilité présente l'avantage (i) de libérer la mémoire qui était utilisée par la machine virtuelle, et (ii) de pouvoir reprendre l'exécution de cette dernière même si la machine physique a été éteinte puis redémarrée, par exemple suite à une opération de maintenance.

Prise d'instantanés

Une forme plus avancée de suspension de l'exécution d'une machine virtuelle est la prise d'instantanés. La différence entre les deux réside dans le fait que l'état du disque est sauvegardé, en plus de l'état du processeur et de la mémoire de la machine virtuelle.

Plusieurs instantanés peuvent être conservés, chacun correspondant à un moment précis de la vie de la machine virtuelle.

Il est alors possible de « remonter le temps », en restaurant la machine virtuelle telle qu'elle était à la date de prise d'un instantané. Ceci autorise par exemple l'usager de la machine virtuelle à la ramener dans un état viable suite à une manipulation qui l'a rendue inexploitable.

Migration

Enfin, la fonctionnalité qui offre le plus de potentiel pour gérer les machines virtuelles dans les infrastructures distribuées est la migration. Migrer une machine virtuelle consiste à la déplacer de sa machine physique d'origine à une machine physique de destination. La migration peut être utilisée pour transférer la charge de travail d'une machine physique à une autre, ou pour libérer une machine physique de ses machines virtuelles avant d'effectuer une opération de maintenance dessus.

La migration peut être effectuée *à froid* ou *à chaud* [CFH+05]. La migration à froid consiste à (i) suspendre l'exécution de la machine virtuelle, (ii) transférer

l'état de la machine virtuelle sur la machine physique de destination, et enfin (iii) reprendre l'exécution de la machine virtuelle. La migration à chaud aspire en revanche à minimiser la durée pendant laquelle la machine virtuelle est inactive ; pour cela, la majeure partie de l'état de la machine virtuelle est transféré alors que celle-ci est toujours en fonctionnement.

Migrer une machine virtuelle est une opération coûteuse puisqu'elle consomme des ressources processeur et réseau [CFH+05, HLM10]. De plus, il faut que la machine virtuelle ait accès en permanence à son disque dur virtuel ; cet objectif peut être atteint en stockant le disque virtuel sur un support accessible par le réseau aux machines physiques d'origine et de destination ; en revanche, si le disque virtuel est stocké en local sur la machine physique d'origine, il est nécessaire de le transférer sur la machine physique de destination lorsque la machine virtuelle est migrée, ce qui rend l'opération de migration d'autant plus coûteuse.

2.2 Virtualisation et gestion des infrastructures distribuées

La virtualisation est de plus en plus utilisée dans les infrastructures distribuées, étant donné les avantages qu'elle présente aussi bien aux fournisseurs qu'aux utilisateurs de ces infrastructures [FZRL08, Vog08, Eri09, HLM+09, Low09].

2.2.1 Apports de la virtualisation pour la gestion des infrastructures distribuées

Avantages pour les fournisseurs

Mutualisation des ressources La virtualisation permet tout d'abord aux fournisseurs de mutualiser les ressources de leur infrastructure, autrement dit d'héberger des machines virtuelles appartenant à différents utilisateurs sur les mêmes nœuds. Cela est rendu possible grâce aux mécanismes qui permettent de gérer finement l'attribution des ressources aux machines virtuelles (comme les cgroups, que nous avons présentés précédemment), et surtout par l'isolation relativement forte entre les machines virtuelles (en théorie, une machine virtuelle ne peut pas accéder à l'état et aux données d'une autre machine virtuelle).

L'utilisation des ressources peut être optimisée en recourant aux fonctionnalités des hyperviseurs (comme le *ballooning* ou la déduplication mémoire) ou bien en migrant des machines virtuelles pour pour tenir compte des fluctuations de la charge de travail (afin d'équilibrer cette charge de travail, ou bien au contraire de la consolider).

La consolidation permet d'utiliser moins de nœuds, et donc de recourir à une climatisation moins puissante. Ceci entraîne une consommation électrique plus faible (vu qu'il y a moins de nœuds à alimenter et à refroidir), d'où un coût financier moindre. En somme, cela permet de réaliser des économies d'échelle.

Facilitation des opérations de maintenance Par ailleurs, la virtualisation facilite les opérations de maintenance. En effet, si une partie quelconque de l'infrastructure a besoin d'être maintenue, il suffit de déplacer les machines virtuelles concernées

à un autre endroit, l'opération de maintenance étant alors totalement transparente pour les utilisateurs.

Déploiement d'un environnement logiciel minimal Enfin, la virtualisation permet aux fournisseurs d'infrastructure de déployer un environnement logiciel minimal sur chaque nœud, sans avoir à gérer les environnements des utilisateurs. La mise en place de ces environnements est en effet laissée aux utilisateurs.

Avantages pour les utilisateurs

Déploiement d'un environnement logiciel personnalisé Les utilisateurs sont en charge de la personnalisation de leur propre environnement logiciel : ils peuvent installer le système d'exploitation invité et les applications qu'ils veulent sur leur machine virtuelle, ils ne sont donc pas limités par l'environnement logiciel de l'infrastructure distribuée.

Externalisation de l'achat et de la gestion de l'infrastructure Les utilisateurs n'ont plus forcément besoin de disposer de leur propre infrastructure ; ils peuvent dès lors en externaliser l'achat et la gestion, et ainsi se décharger d'une opération complexe et coûteuse.

Mieux, ils ne sont plus cantonnés à une infrastructure dimensionnée une fois pour toutes et difficile à faire évoluer : ils peuvent utiliser l'infrastructure virtualisée en fonction de leurs besoins, qui peuvent évoluer dans le temps. Par exemple, un site Web interne à une entreprise sera principalement utilisé pendant les horaires de travail ; l'infrastructure informatique sera donc plus sollicitée à ce moment là.

Tolérance aux pannes et haute disponibilité Enfin, certaines infrastructures proposent des fonctionnalités de tolérance aux pannes et de haute disponibilité aux utilisateurs.

La tolérance aux pannes garantit aux utilisateurs que leurs données ne seront pas perdues en cas de défaillance de l'infrastructure. Quant à la haute disponibilité, elle assure que les machines virtuelles d'un utilisateur seront restaurées aussi rapidement que possible pour avoir une disponibilité quasiment continue.

2.2.2 Virtualisation et centrales numériques

De par son utilité dans les infrastructures distribuées, la virtualisation a contribué à l'essor des centrales numériques.

Définition 2.7 (Centrale numérique)
Une centrale numérique est une fédération d'infrastructures distribuées présentant deux caractéristiques particulières. Tout d'abord, toutes ses composantes sont détenues par la même institution, institution qui propose ses services en l'échange d'une rémunération. Par ailleurs, une centrale numérique met en œuvre un paradigme de calcul appelé informatique en nuage.

Définition 2.8 (Informatique en nuage)
L'informatique en nuage (ou *Cloud Computing*) est « un paradigme de calcul distribué [...] qui est gouverné par des économies d'échelle ». Dans ce paradigme, « un ensemble de ressources de calcul, de stockage, de plates-formes et de services abstraits, virtualisés et pouvant passer à l'échelle de manière dynamique sont fournies à la demande de consommateurs extérieurs par le biais d'Internet » [FZRL08].

La virtualisation est tout particulièrement utilisée par les centrales de type *Infrastructure as a Service* (IaaS), qui proposent des machines virtuelles « nues » aux utilisateurs, à charge à ces derniers d'installer le système d'exploitation invité et les logiciels qu'ils souhaitent utiliser.

L'*Infrastructure as a Service* peut être utilisée par des centrales numériques proposant des services de plus haut niveau, comme celles de type *Platform as a Service* ou *Software as a Service* (cf. figure 2.4). Les centrales numériques de type *Platform as a Service* (PaaS) fournissent une plate-forme logicielle sur laquelle les utilisateurs développent leurs propres applications. Quant à celles de type *Software as a Service* (SaaS), elles hébergent une ou plusieurs applications qui sont directement utilisables par les internautes, par exemple une messagerie ou une suite bureautique en ligne.

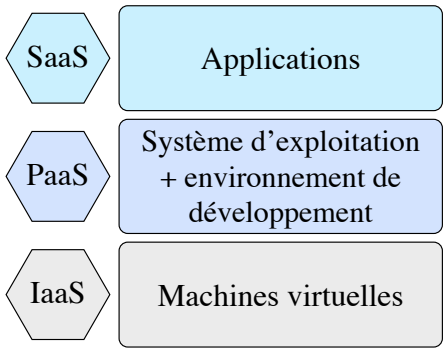


FIGURE 2.4 – Catégories de centrales numériques

Amazon Web Services [aws13], Google App Engine [gae13], Microsoft Windows Azure [azu13], Force.com [for13] et Salesforce.com [sal13] sont des exemples de centrales numériques (cf. figure 2.5 pour une chronologie de leur apparition).

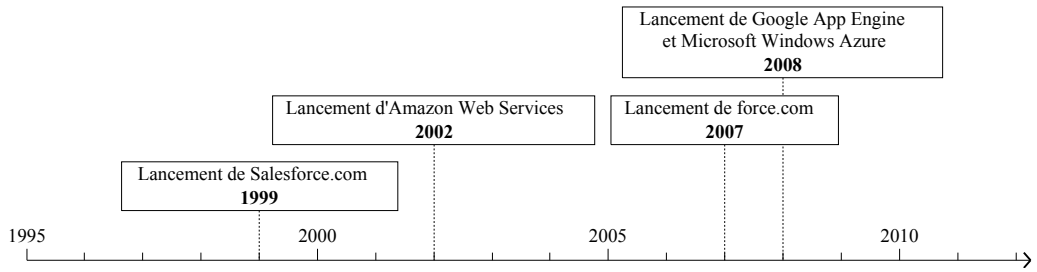


FIGURE 2.5 – Chronologie d'apparition de quelques centrales numériques

2.3 Conclusion

Dans ce chapitre, nous avons présenté les concepts fondamentaux relatifs à la virtualisation, ainsi que leur intérêt dans le cadre de la gestion des infrastructures distribuées.

Nous avons tout d'abord défini le concept de machine virtuelle et identifié les différences entre les machines virtuelles système et applicative.

Nous avons par la suite étudié les hyperviseurs, qui jouent le rôle d'intermédiaire entre les machines virtuelles et physiques. Nous avons vu que ce rôle peut s'apparenter à de la traduction d'instructions, à de l'agrégation ou encore à du partitionnement de ressources. Nous avons remarqué que les hyperviseurs les plus récents avaient tendance à combiner les différentes techniques de virtualisation à leur disposition (l'émulation, la paravirtualisation et la virtualisation matérielle) afin de faciliter le travail des développeurs et de proposer de meilleures performances aux utilisateurs. Nous avons également dressé une liste de fonctionnalités proposées par les hyperviseurs qui sont particulièrement intéressantes dans le cadre d'une infrastructure distribuée, notamment :

- la limitation des ressources utilisées par une machine virtuelle, qui permet par exemple de garantir un partage équitable de ces ressources ;
- l'optimisation de la consommation mémoire, qui permet de démarrer plus de machines virtuelles et d'éviter les situations de pénurie ;
- la suspension et la reprise de l'exécution d'une machine virtuelle ;
- la prise d'instantanés, qui permet de sauvegarder l'état d'une machine virtuelle et de le restaurer ultérieurement ;
- la migration d'une machine physique à une autre, qui permet de transférer la charge de travail et de faciliter les opérations de maintenance.

Nous nous sommes ensuite penchés sur les avantages liés à l'utilisation de la virtualisation dans les infrastructures distribuées. La virtualisation permet ainsi aux fournisseurs d'infrastructure distribuée de mutualiser les ressources, de réaliser des opérations de maintenance plus facilement, ou encore de n'avoir à déployer qu'un environnement logiciel minimal. Du côté des utilisateurs, la virtualisation leur permet de déployer un environnement logiciel adapté à leurs besoins, d'externaliser l'achat et la gestion de ces infrastructures, et de bénéficier des fonctionnalités de haute disponibilité et de tolérance aux pannes. Nous avons enfin souligné le fait que la virtualisation avait contribué à l'essor des centrales numériques de type *Infrastructure as a Service* et de l'informatique en nuage.

Dans le chapitre suivant, nous étudierons les logiciels de gestion d'infrastructure virtualisée qui sont utilisés en production.

3

Solutions logicielles en production pour la gestion des infrastructures virtualisées

La virtualisation système est de plus en plus utilisée dans les infrastructures distribuées depuis la fin des années 2000, du fait des avantages qu'elle apporte aussi bien aux fournisseurs qu'aux utilisateurs de ces infrastructures.

Cet engouement, qui est valable à la fois dans les milieux académique et industriel, a entraîné l'apparition d'un grand nombre de solutions logicielles qui sont dédiées à la gestion des infrastructures virtualisées.

Dans ce chapitre, nous étudions les principaux logiciels de gestion d'infrastructure virtualisée (LGIV) qui sont déployés en production. Nous nous concentrons plus particulièrement sur leur architecture, les fonctionnalités majeures qu'ils proposent, et leurs limitations.

Sommaire

3.1 Présentation des logiciels de gestion d'infrastructure virtualisée

3.1.1 Généralités

Les logiciels de gestion d'infrastructure virtualisée s'appuient sur des concepts très similaires à ceux présentés au chapitre 1. Comme dans le cas d'une infrastructure non virtualisée, un utilisateur doit pouvoir se connecter au système, via un nœud d'accès. Ensuite, avant de pouvoir traiter des tâches, il lui faut créer au moins une machine virtuelle. Pour faciliter et accélérer cette démarche, le logiciel de gestion met généralement à sa disposition des modèles (ou *templates*), autrement dit des machines virtuelles préconfigurées avec un système d'exploitation invité et une certaine capacité en ressources processeur, mémoire, disque dur et réseau. Une fois le modèle choisi, l'ordonnanceur recherche des ressources disponibles en quantité suffisante pour créer la machine virtuelle ; il décide notamment (i) du support de stockage sur lequel copier l'image disque de la machine virtuelle et (ii) du nœud de calcul sur lequel démarrer cette dernière.

3.1.2 Classification

Les logiciels de gestion d'infrastructure virtualisée peuvent être regroupés en plusieurs catégories, suivant qu'ils ont été conçus dans le milieu académique ou industriel et qu'ils sont maintenus par une communauté d'utilisateurs ou non.

Certains logiciels de gestion sont le produit de la recherche académique, comme Eucalyptus [NWG+09, Euc12] (créé à l'université de Californie et commercialisé par Eucalyptus Systems), Nimbus [nim13] (conçu à l'université de Chicago) et OpenNebula [SMLF09, ope13] (conçu à l'université de Madrid et soutenu par la société C12G Labs, avec des contributions de plusieurs partenaires). Nous nous référerons ultérieurement à ces solutions sous l'appellation *solutions académiques*.

D'autres logiciels de gestion ont été élaborés et sont développés par une seule entreprise, comme Microsoft *System Center Virtual Machine Manager* (SCVMM) [Mic12], VMware vSphere [VMw11] et vCloud [VMw10] (vCloud apporte des fonctionnalités supplémentaires à vSphere), et Citrix XenServer [Cit12]. Nous désignerons par la suite ces solutions via l'expression *solutions propriétaires*.

Les logiciels de gestion restants sont issus d'entreprises et sont désormais soutenus par une communauté de développeurs, à l'instar de CloudStack [Apa12] (produit par Citrix) et d'OpenStack [Ope12] (fruit d'une collaboration entre la NASA et Rackspace). Ces solutions seront désormais qualifiées de *solutions communautaires*.

3.2 Organisation des ressources

Comme pour les solutions logicielles utilisées traditionnellement afin de gérer les infrastructures distribuées (cf. chapitre 1), les LGIV sont chargés d'exploiter deux grands types de ressources : les ressources de calcul et les ressources de stockage.

3.2.1 Ressources de calcul

Bien que la terminologie varie fortement d'un LGIV à l'autre, la façon d'organiser les ressources de calcul reste relativement similaire.

Nœuds de calcul et hyperviseurs supportés

La plus petite granularité de travail est le nœud de calcul ; chaque nœud de calcul héberge un hyperviseur.

Les solutions académiques et communautaires sont assez tolérantes en matière d'hyperviseurs ; elles sont en effet capables de gérer des nœuds sur lesquels sont installés KVM, Xen (ou ses dérivés XenServer et Xen Cloud Platform), ESX(i), ou même Hyper-V dans le cas d'OpenStack. Nimbus est la seule exception, puisque seuls KVM et Xen sont reconnus.

A contrario, les solutions propriétaires privilégient généralement l'hyperviseur conçu par la même entreprise. Ainsi, VMware vCloud et vSphere tirent parti des fonctionnalités de ESXi, et l'écosystème Citrix XenServer n'accepte que des nœuds tournant sous l'hyperviseur éponyme. En revanche, Microsoft SCVMM s'accommode aussi bien de Hyper-V que de ESX ou de XenServer.

Unité de regroupement

La plus petite unité de regroupement, qui offre également le plus de flexibilité dans la gestion des nœuds de calcul, équivaut à un ensemble (de taille arbitraire) de nœuds appartenant à une même grappe. Cette unité est nommée *VMM pool* pour Nimbus, *virtual datacenter* pour OpenNebula (avec l'extension oZones), *host groups* pour SCVMM, et *resource pool* pour vCloud/vSphere, XenServer et OpenStack.

La grappe (ou *cluster*) est l'autre unité de gestion couramment rencontrée, notamment avec Eucalyptus, OpenNebula et CloudStack.

Multiples des unités de regroupement

Le premier multiple de ces unités correspond généralement à une fraction d'un centre de données et de calculs. Cela s'appelle *availability zone* pour Nimbus et OpenStack, *private cloud* pour SCVMM, *virtual datacenter* pour vCloud, et *pod* pour CloudStack.

Le plus grand multiple est la *zone*, typiquement un centre de données et de calculs. Ce concept apparaît dans OpenNebula (avec l'extension oZones) et CloudStack, et est en cours de développement pour OpenStack. La gestion des *zones* peut avoir plusieurs finalités : (i) proposer une isolation forte entre utilisateurs de deux *zones* différentes, (ii) diminuer le temps d'accès en aiguillant les utilisateurs vers la *zone* qui est la plus proche d'eux géographiquement, ou encore (iii) mettre en place une *zone* de secours qui est capable de prendre le relais en cas de défaillance de la *zone* principale.

3.2.2 Ressources de stockage

L'organisation des ressources de stockage suit une approche similaire à celle des ressources de calcul.

Stockage local des nœuds de calcul

Certaines solutions font appel au stockage local des nœuds de calcul pour conserver tout ou partie du contenu des disques durs des machines virtuelles. L'utilisation du stockage local permet de garantir un niveau de performance élevé.

Nimbus stocke l'intégralité des images disque des machines virtuelles sur les nœuds de calcul. OpenNebula et CloudStack peuvent également être configurés pour fonctionner de cette manière.

En revanche, Eucalyptus et OpenStack ne stockent que les partitions racines des machines virtuelles sur les nœuds de calcul, chaque partition racine contenant le système d'exploitation invité et la plupart des applications d'une machine virtuelle.

Stockage partagé

En complément ou en remplacement du stockage local, nombre de solutions sont conçues pour tirer parti d'un espace de stockage partagé par les nœuds de calcul d'une même unité. De même que précédemment, cet espace de stockage sert à conserver une partie ou la totalité des images disque des machines virtuelles.

Eucalyptus et OpenStack stockent exclusivement les données utilisateurs persistantes des machines virtuelles sur un support partagé. Eucalyptus utilise pour cela des *storage controllers* (chacun d'entre eux étant associé à un *cluster*), tandis qu'OpenStack se sert de Cinder.

Les autres solutions recourant à du stockage partagé s'en servent pour conserver l'intégralité des images disque des machines virtuelles, ainsi que les instantanés (à l'exception de CloudStack qui garde les instantanés ailleurs, comme nous le verrons par la suite). Cela permet notamment de migrer à chaud des machines virtuelles entre des nœuds ayant accès au même système de stockage partagé. Pour cela, SCVMM utilise des *storage pools* (adjoints aux *host groups*), vCloud/vSphere des *datastores* (assignés aux *resource pools*), et XenServer des *storage repositories* (alliés aux *resource pools*). Si OpenNebula et CloudStack ont été configurés afin de tirer parti d'un stockage partagé, ils utilisent respectivement des *datastores* et des *primary storages* associés aux *clusters*.

Stockage secondaire

En plus du stockage local et/ou partagé, un stockage secondaire est utilisé à l'échelle des multiples des unités de regroupement des nœuds de calcul, afin de conserver des données qui ont une longue durée de vie et qui n'ont pas besoin d'être accédées très rapidement, notamment :

- les instantanés, quand ils ne sont pas stockés sur un support partagé ;
- les modèles de machines virtuelles ;
- éventuellement d'autres types de données.

Eucalyptus se sert de Walrus comme stockage secondaire, Nimbus de Cumulus, OpenNebula d'un *template repository*, SCVMM d'un *VMM library*, vCloud de *datastores* spécifiques, XenServer de *storage repositories* particuliers, CloudStack d'un *secondary storage*, et OpenStack de Glance (pour les modèles et les instantanés) et de Swift (pour tout autre type de données).

Les informations relatives à l'organisation des ressources physiques sont récapitulées dans le tableau 3.1.

Nom	Hyperviseurs supportés	Ressources de calcul	Ressources de stockage
Eucalyptus 3.1.1	*ESXi *KVM *Xen	*clusters* (aussi appelés *availability zones*)	*local (partitions racines) **storage controllers* (stockage persistant des VMs) *Walrus (modèles, instantanés)
Nimbus 2.10	*KVM *Xen	**vmm pools* **availability zones*	*local (images disque) *Cumulus (modèles, instantanés)
OpenNebula 3.8	*ESX *KVM *Xen	**clusters* **virtual datacenters* (1 VDC = partie d'un *cluster*) **zones* (1 *zone* = 1 centre de données)	*local (images disque des VMs actives, si copie en local à l'instanciation des VMs) *local du *frontend* (images disque des VM arrêtées, si copie en local à l'instanciation des VMs) **datastores* (images disque, instantanés, iso) **template repository* (modèles)
SCVMM 2012	*ESX *Hyper-V *XenServer	**host groups* **private clouds*	**storage pools* (images disque, instantanés) **vmm library* (modèles, iso)
vCloud / vSphere 5.1	ESXi	**resource pools* (1 RP = partie d'une grappe ; grappe max 32 nœuds) *vCloud *provider virtual datacenters* (1 PVDC = 1 *cluster* vSphere) *vCloud *organization VDC* (1 OVDC = partie d'un PVDC)	**datastores* (images disque, instantanés, iso, modèles vCloud) *stockage de la machine client ou URL (modèles vSphere)
XenServer 6.0	XenServer	*resource pools* (1 RP = partie d'une grappe ; grappe max 16 nœuds)	*storage repositories* (images disque, instantanés, iso, modèles)
CloudStack 4.0.0	*ESX *KVM *Xen, XenServer	**clusters* **pods* **zones* (typiquement un centre de données)	*local (images disque, si configuré ainsi) **primary storage* (*cluster* ; images disque, si configuré ainsi) **secondary storage* (*zone* ; modèles, iso et instantanés)
OpenStack Folsom	*ESXi *Hyper-V *KVM *Xen, XenServer, Xen Cloud Platform	**resource pools* **availability zones* **zones* (travail en cours)	*local (partitions racines) *Cinder, anciennement Nova *volume workers* (stockage persistant des VMs) *Glance (modèles, iso, instantanés) *Swift (tout autre type de données)

TABLEAU 3.1 – Organisation des ressources physiques

3.3 Ordonnancement lié aux ressources de calcul

La répartition des machines virtuelles sur les ressources physiques, et plus particulièrement les nœuds de calcul, est effectuée par des ordonnanceurs dédiés dont nous allons maintenant étudier les caractéristiques.

3.3.1 Architecture des ordonnanceurs

La première caractéristique des ordonnanceurs des solutions étudiées est leur architecture centralisée ou hiérarchique.

Architecture centralisée

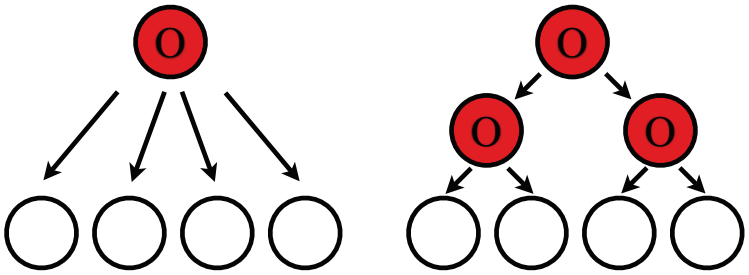
Définition 3.1 (Ordonnanceur centralisé)
Dans le cas d'une architecture centralisée, l'ordonnanceur est installé sur un unique nœud.

La grande majorité des ordonnanceurs des solutions étudiées a recours à une architecture centralisée (cf. figure 3.1(a)) ; seuls ceux d'Eucalyptus et de XenServer suivent une approche hiérarchique.

Architecture hiérarchique

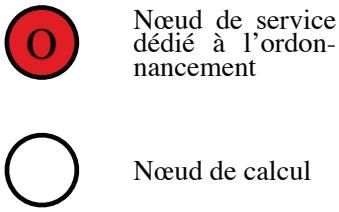
Définition 3.2 (Ordonnanceur hiérarchique)
Dans le cas d'une architecture hiérarchique, l'ordonnanceur est formé de plusieurs composantes organisées de manière hiérarchique. Chaque composante est chargée de gérer un sous ensemble de l'infrastructure.

Les hiérarchies mises en place par Eucalyptus et XenServer comportent deux échelons (cf. figure 3.1(b)) : le premier échelon correspond à l'ordonnancement au niveau d'un *cluster* ou d'un *resource pool* ; le deuxième échelon concerne quant à lui la gestion de l'infrastructure tout entière.



(a) Ordonnanceur centralisé

(b) Ordonnanceur hiérarchique



(c) Légende

FIGURE 3.1 – Comparaison entre un ordonnanceur centralisé et hiérarchique

3.3.2 Facteurs déclenchant un ordonnancement

Indépendamment de l'architecture, plusieurs facteurs sont susceptibles de déclencher un ordonnancement.

Création d'une nouvelle machine virtuelle

Le premier facteur est la création d'une nouvelle machine virtuelle par un utilisateur ; l'ordonnanceur recherche alors des ressources disponibles pour pouvoir la démarrer. Ce facteur est géré par l'ensemble des solutions étudiées.

Optimisation périodique ou sur demande de l'utilisation des ressources

Au cours de l'existence de la machine virtuelle, il peut être nécessaire de la déplacer pour optimiser l'utilisation des ressources de l'infrastructure.

Cette optimisation est susceptible d'être déclenchée sur demande d'un administrateur (comme c'est le cas pour OpenNebula et SCVMM), et/ou d'être effectuée de manière périodique par l'ordonnanceur (comme c'est le cas pour SCVMM, vCloud/vSphere et XenServer). Il est à noter que la durée d'une période varie fortement d'une solution à l'autre : elle peut être configurée par l'administrateur pour prendre toute valeur entre dix minutes et vingt-quatre heures pour SCVMM, et entre une minute et une heure pour vCloud/vSphere ; en revanche, elle est fixée à deux minutes pour XenServer.

Maintenance d'un nœud

À un autre moment du cycle de vie de la machine virtuelle, un administrateur de l'infrastructure peut vouloir réaliser des opérations de maintenance sur le nœud qui l'héberge, par exemple pour mettre à jour des logiciels ou changer un composant matériel défectueux.

Afin d'éviter tout impact négatif sur le fonctionnement de la machine virtuelle, l'administrateur a le choix (i) de l'arrêter pendant toute la durée de la maintenance, ou (ii) de la migrer sur un autre nœud. Ce processus d'évacuation de la machine virtuelle est susceptible d'être automatisé par l'ordonnanceur. C'est d'ailleurs le cas pour OpenNebula, SCVMM, vCloud/vSphere, XenServer et CloudStack.

Défaillance d'une machine virtuelle

Enfin, la machine virtuelle est également soumise au risque de panne logicielle ou matérielle. En cas de défaillance d'une machine virtuelle signalée comme devant être hautement disponible, l'ordonnanceur est averti et peut alors redémarrer cette machine virtuelle.

Cette fonctionnalité est proposée par OpenNebula, SCVMM, vCloud/vSphere, XenServer et CloudStack. Elle est en cours de développement pour OpenStack.

3.3.3 Politiques d'ordonnancement

Une fois qu'un ordonnancement est déclenché, celui-ci se conforme à une certaine politique d'ordonnancement, qui définit quelles ressources assigner en priorité à un ensemble de machines virtuelles donné.

Tous les logiciels de gestion étudiés proposent leur(s) propre(s) politique(s) d'ordonnancement. Seul Nimbus offre également la possibilité d'utiliser un logiciel externe, de type ordonnanceur par lots, pour gérer les ressources.

Premier ajustement

La politique de premier ajustement (ou *first fit*) est la plus simple, puisqu'elle a pour objectif de sélectionner le premier nœud avec suffisamment de ressources disponibles pour héberger une machine virtuelle nouvellement créée.

Cette politique est mise en œuvre par Eucalyptus et CloudStack.

Aléatoire

Une autre approche, dérivée de la politique de premier ajustement, consiste à lister tous les nœuds pouvant héberger la nouvelle machine virtuelle et à en choisir un de manière aléatoire.

Cette approche est proposée par OpenStack.

Équilibrage de charge

Plutôt que de sélectionner un nœud au hasard, il peut être judicieux de chercher à équilibrer la charge de travail sur l'ensemble des nœuds, de façon à ce que les machines virtuelles bénéficient des meilleures performances possibles. Dans ce cas, les nœuds les moins chargés sont utilisés en priorité pour accueillir des machines virtuelles.

La charge de travail peut être définie de différentes manières, notamment en fonction :

- du nombre de machines virtuelles hébergées par un nœud ; c'est sur cette définition que repose l'algorithme *striping* d'OpenNebula ;
- du taux d'utilisation de la mémoire vive d'un nœud ; l'algorithme *round robin* de Nimbus et l'ordonnanceur *filter* d'OpenStack (lorsqu'il est configuré pour faire de l'équilibrage de charge) s'appuient sur cette définition ;
- du taux d'utilisation du processeur d'un nœud ; l'algorithme *load-aware* d'OpenNebula utilise cette définition ;
- d'une combinaison du taux d'utilisation de plusieurs ressources d'un nœud, par exemple (i) du processeur et de la mémoire vive dans le cas de l'algorithme *distributed resource scheduling* de vSphere, ou (ii) du processeur, de la mémoire vive, du disque dur et du réseau dans le cas des algorithmes *dynamic optimization* de SCVMM et *performance* de XenServer.

Consolidation

À l'inverse de l'équilibrage de charge, la consolidation vise à utiliser au maximum les ressources d'un nombre restreint de nœuds.

De la même manière que pour l'équilibrage de charge, un algorithme de consolidation peut considérer différents critères :

- le nombre de machines virtuelles hébergées par un nœud ; c'est le cas de l'algorithme *packing* d'OpenNebula ;

– le taux d'utilisation de la mémoire vive d'un nœud ; c'est le cas de l'algorithme *greedy* de Nimbus et de l'ordonnanceur *filter* d'OpenStack (lorsqu'il est configuré pour faire de la consolidation) ;
– une combinaison du taux d'utilisation de différentes ressources d'un nœud, comme (i) le processeur et la mémoire vive pour l'algorithme *distributed power management* de vSphere, ou (ii) le processeur, la mémoire vive, le disque dur et le réseau pour les algorithmes *power optimization* de SCVMM et *density* de XenServer.

Un algorithme de consolidation dynamique peut être utilisé pour libérer des nœuds de toutes leurs machines virtuelles. Ces nœuds peuvent alors être éteints dans un souci d'économie d'énergie. Ils ne sont rallumés que lorsqu'ils sont de nouveau nécessaires à l'hébergement de machines virtuelles. Cette fonctionnalité est proposée aussi bien par SCVMM que par vSphere et XenServer.

Affinités et antagonismes

Enfin, une dernière politique d'ordonnancement, qui peut être utilisée en complément d'une des politiques présentées précédemment, tire profit de la définition d'affinités et d'antagonismes.

Ces affinités ou antagonismes peuvent être défini(e)s (i) au sein d'un groupe de machines virtuelles ou (ii) entre un groupe de machines virtuelles et un groupe de nœuds.

Dans le cas où les affinités/antagonismes sont exprimé(e)s sur un groupe de machines virtuelles, l'objectif est :
– de regrouper sur un même nœud les machines virtuelles ayant des affinités (par exemple pour leur permettre de communiquer plus rapidement) ;
– de faire en sorte que deux machines virtuelles antagonistes ne soient jamais hébergées sur le même nœud (par exemple pour éviter que toutes les machines virtuelles hébergeant un même site Web tombent simultanément en cas de panne d'un nœud).

Dans le cas où les affinités/antagonismes sont spécifié(e)s entre un groupe de machines virtuelles et un groupe de nœuds, le but est :
– de privilégier, voire de forcer, l'utilisation de ces nœuds pour le placement des machines virtuelles avec lesquelles ils ont des affinités ;
– d'interdire le placement des machines virtuelles sur les nœuds avec lesquels elles ont des antagonismes.

Tous ces types d'affinités/antagonismes sont disponibles sous vSphere. OpenStack ne propose quant à lui que les affinités/antagonismes entre machines virtuelles.

Les informations relatives à l'ordonnancement sont synthétisées dans le tableau 3.2.

3.4 Avantages

En dehors de l'ordonnancement des machines virtuelles, les solutions étudiées présentent un certain nombre d'intérêts, aussi bien pour les fournisseurs d'infrastructure que pour les utilisateurs. Nous nous concentrerons sur les aspects relatifs aux interfaces, à l'isolation entre les utilisateurs, au passage à l'échelle, et à la tolérance aux pannes.

Nom	Type d'ordonnancement	Déclenchement ordonnancement	Politiques d'ordonnancement	Ressources discriminantes pour l'ordonnancement
Eucalyptus 3.1.1	statique	création d'une VM	premier ajustement (*first fit*)	-
Nimbus 2.10	statique	création d'une VM	*Nimbus (*greedy* = consolidation mémoire ; *round robin* = équilibrage de charge mémoire) **Local Resource Manager* (ex : Torque)	mémoire
OpenNebula 3.8	statique (principalement)	*création d'une VM *demande ré-ordonnancement explicite d'une VM *maintenance *défaillance d'une VM	**packing* : choisit les nœuds avec le plus de VMs **striping* : choisit les nœuds avec le moins de VMs **load-aware* : équilibrage de charge processeur	aucune ou processeur
SCVMM 2012	dynamique	*création d'une VM *période (10 min $\leq$ période $\leq$ 24 h) *à la demande, sur un *host group* *maintenance *défaillance d'une VM	**dynamic optimization* (équilibrage de charge) **power optimization* (consolidation avec extinction des nœuds ; fonctionnalité optionnelle de *dyn. optim.*)	*processeur *mémoire *disque (E/S) *réseau (E/S)
vCloud / vSphere 5.1	dynamique	*création d'une VM *période (1 min $\leq$ période $\leq$ 1 h, 5 minutes par défaut) *maintenance *défaillance d'une VM	**distributed resource scheduling* (équilibrage de charge) **distributed power management* (consolidation avec extinction des nœuds) *affinités et antagonismes (entre VMs, ou entre un groupe de VMs et un groupe de nœuds)	*processeur *mémoire
XenServer 6.0	dynamique	*création d'une VM *période (2 minutes) *maintenance *défaillance d'une VM	**performance* (équilibrage de charge) **density* (consolidation avec extinction des nœuds)	*processeur *mémoire *disque (E/S) *réseau (E/S)
CloudStack 4.0.0	statique (principalement)	*création d'une VM *maintenance *défaillance d'une VM	premier ajustement (*first fit*)	-
OpenStack Folsom	statique	*création d'une VM	**filter* : choisit le nœud ayant le meilleur score (en pratique, équilibrage de charge ou consolidation mémoire) **chance* : choisit un nœud au hasard (avec assez de capacité) **simple* (équilibrage de charge) : choisit le nœud le moins chargé *affinités et antagonismes (entre VMs)	mémoire

TABLEAU 3.2 – Ordonnancement des machines virtuelles sur les ressources de calcul

3.4.1 Interfaces de programmation et interfaces utilisateurs

Les fonctionnalités de chaque logiciel de gestion sont accessibles au travers d'une ou plusieurs interfaces de programmation (*Application Programming Interface*, ou API). Notons que les solutions académiques et communautaires sont partiellement compatibles avec l'interface de programmation du service *Elastic Compute Cloud* (EC2) de la centrale numérique d'Amazon. Cela présente l'avantage pour les utilisateurs d'EC2 qui souhaiteraient transférer leurs machines virtuelles sur une infrastructure privée de continuer à utiliser les mêmes outils. À l'inverse, ces utilisateurs pourraient tester les principales fonctionnalités d'EC2 sur leur propre infrastructure avant de basculer sur le service d'Amazon. Il est important de rappeler que l'interface EC2 est un standard *de facto*, étant donné le poids d'Amazon sur le marché de l'*Infrastructure as a Service*.

Les outils pour accéder à EC2, ou à toute autre infrastructure virtualisée, peuvent se présenter sous la forme d'applications utilisables en ligne de commande.

Afin de faciliter leur utilisation par des non informaticiens, ces infrastructures peuvent également être accédées via des interfaces graphiques. Ces interfaces se présentent généralement sous la forme d'un portail Web (c'est le cas pour toutes les solutions étudiées), mais parfois aussi sous l'apparence d'un client lourd (autrement dit, un programme installé sur l'ordinateur de l'utilisateur).

3.4.2 Isolation entre utilisateurs

Les fonctionnalités et les ressources accessibles au travers des interfaces diffèrent en fonction de l'utilisateur et des droits qu'il détient.

Groupes et quotas

Les utilisateurs peuvent être réunis au sein de groupes.

Suivant la solution utilisée, il est alors possible de définir, pour un groupe ou un utilisateur donné, sur quels nœuds il peut créer des machines virtuelles, combien de ces machines virtuelles il peut créer, et à quelle quantité et types de ressources processeur, mémoire, disque et réseau il a droit.

Ces quotas permettent de protéger les usagers en empêchant un utilisateur ou un groupe d'utilisateurs de consommer une trop grande quantité de ressources.

Isolation réseau

Une autre manière de protéger les utilisateurs les uns des autres consiste à isoler le trafic réseau en provenance et à destination des machines virtuelles. Les solutions étudiées proposent deux moyens en particulier pour atteindre cet objectif.

Le premier est de configurer des réseaux locaux virtuels (*Virtual Local Area Network*, ou VLAN) afin de séparer le trafic réseau en provenance et à destination des différentes machines virtuelles ; chaque VLAN est l'équivalent virtuel d'un réseau physique dédié. Les VLAN ne sont cependant pas suffisants pour isoler des machines virtuelles qui sont connectées à Internet, puisque le trafic réseau d'un VLAN peut transiter par Internet afin d'entrer dans un autre VLAN.

Les limites des VLAN sont compensées par le filtrage du trafic réseau. Le but est d'empêcher que certains types de trafic réseau entrent dans (voire sortent d') une machine virtuelle ou un groupe de machines virtuelles. Eucalyptus, OpenNebula et

OpenStack ne permettent de filtrer que le trafic réseau entrant ; vCloud/vSphere (via vShield Zones), XenServer (via Distributed vSwitch) et CloudStack offrent en revanche la possibilité de filtrer aussi bien le trafic réseau entrant que sortant.

Les informations relatives aux interfaces et à l'isolation réseau entre utilisateurs sont récapitulées dans le tableau 3.3.

	Interfaces		**Isolation**	
Nom	**Support API EC2**	**Portail web *self service***	**VLAN**	**Filtrage du trafic réseau des VMs**
Eucalyptus 3.1.1	oui	oui	oui	*trafic entrant *pour un groupe de VMs (*security group*)
Nimbus 2.10	oui	oui	?	?
OpenNebula 3.8	oui	oui	oui	*listes blanches et noires de ports, trafic entrant *pour chaque VM créée à partir du modèle contenant ces listes
SCVMM 2012	non	oui	oui	?
vCloud / vSphere 5.1	non	oui	oui	*trafic entrant et sortant *pour toutes les VMs du centre de données, d'une grappe, ou d'un réseau *via vShield Zones
XenServer 6.0	non	oui	oui	*trafic entrant et sortant *pour une VM, toutes les VMs d'un réseau, d'un *pool*, ou du centre de données *via Distributed vSwitch
CloudStack 4.0.0	oui	oui	oui	*trafic entrant et sortant *pour un groupe de VMs (*security group*)
OpenStack Folsom	oui	oui	oui	*trafic entrant *pour un groupe de VMs (*security group*)

TABLEAU 3.3 – Interfaces proposées et isolation réseau entre utilisateurs

3.4.3 Passage à l'échelle

La configuration du réseau est une opération clé, puisqu'en plus de renforcer l'isolation entre les utilisateurs, elle permet également de faciliter le passage à l'échelle de leurs applications. Une application passant à l'échelle est capable d'absorber un accroissement substantiel de sa charge de travail sans que cela ne dégrade notablement la qualité du service fourni.

Équilibrage de charge réseau

Une première manière de faciliter ce passage à l'échelle, lorsqu'un site Web à fort trafic est concerné, consiste à installer ce site sur un groupe de machines virtuelles et à répartir le trafic réseau aussi équitablement que possible entre celles-ci.

Plusieurs politiques d'équilibrage de charge réseau coexistent au sein des solutions étudiées :

- la politique du tourniquet (*round robin*) assigne toute nouvelle connexion (au site Web) à une machine virtuelle différente du groupe ; elle est mise en œuvre par SCVMM, vCloud/vSphere, CloudStack et OpenStack ;
- la politique de moindre connexion (*least connection*) aiguille toute nouvelle connexion vers la machine virtuelle qui a le moins de connexions à gérer ; cette politique est fournie avec CloudStack et OpenStack ;
- la politique de l'adresse IP d'origine (*source IP* ou *hash IP*) dirige toute nouvelle connexion vers une machine virtuelle donnée en fonction de la valeur de l'adresse IP depuis laquelle est effectuée cette connexion ; elle est proposée par vCloud/vSphere et CloudStack.

Dimensionnement automatique des applications (*auto scaling*)

L'équilibrage de charge réseau prend tout son intérêt lorsqu'il est couplé au dimensionnement automatique des applications (*auto scaling*). Autrement dit, lorsque le logiciel de gestion d'infrastructure virtualisée est capable de démarrer ou d'éteindre des machines virtuelles faisant fonctionner une application, en fonction de la charge de travail qu'elle doit absorber à un instant donné.

Cette fonctionnalité est proposée par Nimbus, mais uniquement pour les utilisateurs de la grille FutureGrid. Par ailleurs, elle est en cours de développement pour CloudStack et OpenStack.

3.4.4 Haute disponibilité et tolérance aux pannes

Les avantages décrits jusqu'à maintenant présenteraient un intérêt limité si les machines virtuelles ou le logiciel de gestion cessaient de fonctionner à la première défaillance d'un nœud. Heureusement, plusieurs mécanismes sont mis en place par la plupart des solutions étudiées afin de limiter les conséquences des pannes matérielles. Ces solutions font tout particulièrement appel à la réplication et au redémarrage automatique de services.

Définitions

La réplication, telle qu'elle est mise en œuvre dans les solutions étudiées, permet de rétablir un service de manière presque instantanée en cas de panne, tout en limitant au maximum les pertes de données. Typiquement, un service de secours se tient prêt à prendre le relais du service principal ; dès que la défaillance du service principal est détectée, le service de secours s'active.

Le redémarrage automatique permet quant à lui de rétablir le service rapidement, mais pas autant que la réplication. Lorsqu'un service tombe, il est redémarré, potentiellement sur un nœud différent ; il faut alors attendre durant toute la phase d'initialisation avant qu'il soit de nouveau opérationnel.

Haute disponibilité et tolérance aux pannes des machines virtuelles

La réplication des machines virtuelles n'est proposée que par vCloud/vSphere. Une machine virtuelle « miroir » est créée sur un nœud différent de celui hébergeant la machine virtuelle d'origine (qui est spécifiée comme devant être tolérante aux

pannes) ; tout ce qui se produit sur la machine d'origine est répliqué sur la machine miroir ; si la machine d'origine tombe, la machine miroir prend immédiatement le relais.

En revanche, la fonctionnalité de redémarrage automatique des machines virtuelles est utilisable avec toutes les solutions étudiées, à l'exception d'Eucalyptus et d'OpenStack, sachant qu'elle est en cours de développement pour cette dernière. En cas de défaillance d'une machine virtuelle marquée comme devant être hautement disponible, celle-ci est automatiquement redémarrée, éventuellement sur un autre nœud que le nœud initial si ce dernier ne fonctionne plus. L'image disque de la machine virtuelle doit généralement être hébergée sur un stockage partagé par les nœuds d'origine et de destination pour pouvoir utiliser cette fonctionnalité.

Haute disponibilité et tolérance aux pannes des composants du logiciel de gestion d'infrastructure

La fonctionnalité de redémarrage automatique des machines virtuelles est intéressante à plus d'un titre, car elle permet de protéger non seulement les machines virtuelles des utilisateurs, mais également certains composants du logiciel de gestion d'infrastructure, si ceux-ci sont installés sur des machines virtuelles. Concernant les autres composants, il faut aviser au cas par cas.

Les bases de données doivent être répliquées, soit via un mécanisme qui est fourni avec le logiciel de gestion des bases de données (comme c'est le cas avec MySQL et Oracle notamment), soit via un mécanisme externe (VMware vCenter Server Heartbeat pour les bases de données Microsoft SQL par exemple).

Enfin, les composants sans état (c'est-à-dire ceux qui peuvent être redémarrés sans qu'il y ait de perte de données), peuvent être soit répliqués soit redémarrés automatiquement.

Reprise d'activité après un sinistre

Les fonctionnalités de haute disponibilité et de tolérance aux pannes exposées précédemment ne sont malheureusement d'aucune utilité en cas de sinistre affectant l'ensemble d'un centre de données et de calculs, comme les coupures d'électricité de longue durée, les catastrophes météorologiques, les incendies, les inondations, les tremblements de terre, ou encore les crashs d'avions.

Certaines solutions permettent néanmoins de faciliter la reprise d'activité en répliquant les données sur un site de secours. C'est notamment le cas de vSphere et de XenServer.

Les informations concernant le passage à l'échelle, la haute disponibilité et la tolérance aux pannes sont reprises dans le tableau 3.4.

3.5 Limites

Malgré tous les avantages découlant de l'utilisation des logiciels de gestion étudiés, ceux-ci présentent un certain nombre de limitations, principalement liées aux ordonnanceurs et aux interfaces.

	Passage à l'échelle		Haute disponibilité		
Nom	**Équilibrage de charge réseau entre VMs**	**Dimensionnement automatique des applications**	**VMs**	**Composants logiciels du gestionnaire**	**Reprise d'activité après sinistre**
Eucalyptus 3.1.1	non	non	non	oui	?
Nimbus 2.10	?	proposé sur FutureGrid via Phantom 0.1	proposée sur FutureGrid via Phantom 0.1	?	?
OpenNebula 3.8	non	non	oui	*réplication MySQL **frontend* : à faire manuellement	?
SCVMM 2012	tourniquet	non	oui	oui	?
vCloud / vSphere 5.1	*tourniquet **hash IP* (via vShield Edge Web LB, port 80 seulement)	?	oui	*vCenter Server : via vCenter Server Heartbeat *vCenter BDD : via Heartbeat (si MS SQL) ou Oracle RAC (si Oracle) *autres BDD : Oracle RAC *vCenter Chargeback : nécessité de répartir les composants *vCloud Director : déployé sous forme de *pool* de VMs HD	oui (via vCenter Site Recovery Manager)
XenServer 6.0	?	?	oui	**pool master*	oui
CloudStack 4.0.0	*tourniquet *moindre connexion *IP d'origine	en cours de développement	oui	oui *support de multiples serveurs de gestion *réplication MySQL	non
OpenStack Folsom	(via Atlas LB) *tourniquet *moindre connexion	prévu via Heat	prévue via Heat	?	?

TABLEAU 3.4 – Passage à l'échelle et tolérance aux pannes

3.5.1 Ordonnancement

Tout d'abord, la moitié des logiciels de gestion étudiés ne permettent pas d'optimiser dynamiquement l'utilisation des ressources. Étant donné que l'activité d'une machine virtuelle est susceptible de fluctuer dans le temps, celle-ci ne consommera pas nécessairement l'intégralité des ressources mises à sa disposition par le nœud qui l'héberge. Concrètement, cela signifie qu'une partie des ressources seront sous-utilisées.

Ensuite, la réactivité des ordonnanceurs permettant de faire de l'optimisation dynamique est perfectible. Il faut en effet attendre jusqu'à deux minutes avec

XenServer, cinq minutes avec vSphere (il est possible de descendre à une minute, mais ce n'est pas conseillé) et même dix minutes avec SCVMM avant qu'un problème de surutilisation d'un nœud soit pris en compte, sans mentionner le temps nécessaire pour corriger le problème. Autrement dit, la qualité du service est dégradée pour une durée plus ou moins importante.

Par ailleurs, le ré-ordonnancement des machines virtuelles en cas de maintenance ou de défaillance gagnerait à être généralisé, respectivement pour faciliter la tâche des administrateurs et pour améliorer la disponibilité des applications des clients.

De plus, les logiciels de gestion (à l'exception de vSphere) imposent d'utiliser un moyen de stockage partagé entre plusieurs nœuds pour pouvoir migrer des machines virtuelles à chaud d'un nœud à l'autre. Or ce stockage partagé peut constituer un goulet d'étranglement si plusieurs machines virtuelles effectuent simultanément de nombreuses opérations de lecture/écriture sur leurs images disque respectives [Kot10] ; autrement dit, ce stockage partagé ne permet pas de passer à l'échelle.

Enfin, les ordonnanceurs de machines virtuelles sont très majoritairement centralisés, ce qui limite là encore leur passage à l'échelle. Autrement dit, ils ne sont pas aptes à ordonnancer un grand nombre de machines virtuelles dans un temps raisonnable, surtout s'ils doivent pour cela prendre en considération les ressources d'un nombre conséquent de nœuds.

3.5.2 Interfaces

Ce problème de centralisation se retrouve dans les interfaces graphiques des logiciels de gestion, qui ne sont pas adaptées à la gestion de milliers de machines virtuelles et de centaines de nœuds [Kot10]. Les administrateurs ont alors tout intérêt à se tourner vers des outils utilisables en ligne de commande.

Par ailleurs, l'interface d'Amazon EC2 n'est que partiellement supportée (voire par supportée du tout) par les logiciels de gestion étudiés. En particulier, le dimensionnement automatique des applications n'est proposé à l'heure actuelle que par Nimbus aux utilisateurs de FutureGrid.

3.6 Conclusion

Dans ce chapitre, nous avons étudié les principaux logiciels de gestion d'infrastructure virtualisée qui sont déployés en production : Eucalyptus [NWG+09], Nimbus [nim13], OpenNebula [SMLF09], SCVMM [Mic12], vCloud [VMw10], vSphere [VMw11], XenServer [Cit12], CloudStack [Apa12] et OpenStack [Ope12].

Nous nous sommes tout d'abord intéressés à la manière dont ces logiciels de gestion organisaient les ressources des infrastructures distribuées. Nous avons constaté à cette occasion que les infrastructures administrées allaient de la grappe au centre de données et de calculs, voire même à la fédération de centres de données et de calculs. Ce découpage permet d'offrir une plus grande flexibilité aux fournisseurs et aux utilisateurs de ces infrastructures.

Nous nous sommes ensuite concentrés sur l'ordonnancement des machines virtuelles. Nous avons vu que les ordonnanceurs étaient majoritairement centralisés, que les machines virtuelles étaient souvent ordonnancées de manière statique, et que les politiques d'ordonnancement les plus fréquentes étaient l'équilibrage de charge et la consolidation.

Nous avons souligné les avantages liés à l'utilisation de la virtualisation, aussi bien pour les utilisateurs que pour les fournisseurs et les administrateurs de l'infrastructure. Nous avons notamment mentionné la possibilité d'isoler les utilisateurs les uns des autres, de faciliter le passage à l'échelle de leurs applications, et de redémarrer automatiquement les machines virtuelles en cas de défaillance, ce qui est essentiel pour les applications qui doivent être disponibles de manière quasi permanente.

Nous avons enfin relevé les limites des logiciels de gestion étudiés, en matière d'interfaces et d'ordonnancement. Nous avons notamment souligné leur nature centralisée, qui limite leur capacité à passer à l'échelle, autrement dit à gérer des infrastructures de grande taille.

Nous nous intéresserons par la suite à l'amélioration du passage à l'échelle des logiciels de gestion d'infrastructure virtualisée, en étudiant la possibilité de décentraliser le traitement de certaines tâches de gestion.

II

Vers une solution logicielle coopérative et décentralisée pour la gestion des infrastructures virtualisées

4

Étude comparative des logiciels de gestion d'infrastructure virtualisée et des systèmes d'exploitation distribués

Au cours des chapitres précédents, nous avons étudié la gestion des infrastructures distribuées avant et après l'essor de la virtualisation système. Nous avons souligné le fait que cet essor avait entraîné l'apparition d'une nouvelle catégorie de logiciels de gestion. Nous avons en outre remarqué que, même si ces logiciels de gestion d'infrastructure virtualisée (LGIV) proposaient des mécanismes évolués permettant une utilisation plus simple et efficace des ressources, ils étaient souvent hautement centralisés, ce qui limitait leur capacité à passer à l'échelle.

L'une des façons de résoudre ce problème est de décentraliser le traitement des tâches de gestion. Or cette problématique de décentralisation a déjà été abordée par les recherches sur les systèmes d'exploitation distribués (SED). Il serait donc intéressant de savoir si les résultats de ces recherches pourraient permettre d'améliorer les LGIV, de la même manière que les recherches sur les systèmes d'exploitation et les hyperviseurs se sont avérées complémentaires à l'échelle d'un nœud isolé [HWF+05, HUL06, REH07]. Dans ce chapitre, nous proposons de répondre à cette interrogation :

- en développant les comparaisons initiées précédemment entre les systèmes d'exploitation et les hyperviseurs à l'échelle d'un nœud isolé [HWF+05, HUL06, REH07] ;
- en étendant cette comparaison au contexte distribué, afin d'analyser si certains concepts mis en œuvre dans les SED peuvent être appliqués aux LGIV.

Cette étude a fait l'objet d'une publication dans une conférence internationale avec comité de lecture [QL11].

Sommaire

4.1 Comparaison dans un contexte local

Dans cette section, nous nous concentrons sur la gestion des ressources locales, en comparant les fonctionnalités offertes par les systèmes d'exploitation généralistes (SEG) et les hyperviseurs en matière de gestion des processus et des machines virtuelles, d'ordonnancement et de gestion de la mémoire.

Afin d'éviter les répétitions, et lorsque la distinction ne sera pas nécessaire, nous utiliserons le terme de *tâche* pour désigner un processus (respectivement une machine virtuelle), et le terme de *gestionnaire* pour faire référence à un SEG (respectivement un hyperviseur).

4.1.1 Cycle de vie des tâches

Un gestionnaire doit gérer les tâches tout au long de leur cycle de vie et leur permettre d'exécuter des instructions privilégiées.

Gestion du cycle de vie des tâches

Un gestionnaire a la capacité de créer de nouvelles tâches en chargeant en mémoire les instructions appropriées à partir d'un support de stockage local ou distant.

Une fois qu'une nouvelle tâche est créée, le gestionnaire l'intègre à l'ensemble des tâches à ordonnancer ; la tâche est exécutée dès qu'elle est sélectionnée par l'ordonnanceur.

Avant qu'elle soit achevée, l'ordonnanceur peut décider de la préempter pour laisser la place à une autre tâche. Cette opération se nomme *changement de contexte*. Elle consiste à sauvegarder le contenu des registres du processeur utilisé par la tâche en cours d'exécution et à restaurer le contenu des registres correspondant à la tâche à exécuter.

Une tâche peut également avoir besoin d'attendre qu'un événement donné se produise (par exemple, qu'une opération de lecture ou d'écriture sur le disque dur se termine) avant de poursuivre son exécution ; l'ordonnanceur la place alors dans une file d'attente.

Une fois qu'une tâche est terminée, le gestionnaire fait en sorte qu'elle ne puisse plus être sélectionnée pour l'ordonnancement et libère les pages mémoire qu'elle utilisait, afin qu'elles puissent être assignées à une autre tâche.

Le cycle de vie d'une tâche est récapitulé sur la figure 4.1.

Exécution d'instructions privilégiées

Au cours de son cycle de vie, une tâche peut avoir besoin d'exécuter des instructions privilégiées ; elle fait alors appel au gestionnaire.

Un processus utilise pour cela un appel système à destination du SEG.

Dans le cas de l'émulation ou de la virtualisation matérielle, un SEG invité exécute ses instructions normalement ; en revanche, dans le cas de la paravirtualisation, un SEG invité utilise un *hypercall*, c'est-à-dire un appel système à destination de l'hyperviseur.

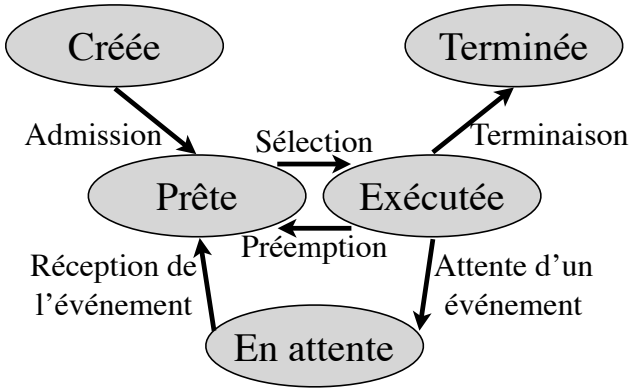


FIGURE 4.1 – Cycle de vie d'une tâche [SG98]

Bilan

Les SEG et les hyperviseurs proposent les mêmes fonctionnalités essentielles (création, changement de contexte, destruction, exécution d'instructions privilégiées) pour gérer respectivement des processus et des machines virtuelles.

4.1.2 Ordonnancement

Étant donné que toutes les tâches ne peuvent pas être traitées de manière simultanée par la même unité de calcul, le gestionnaire a la responsabilité de les ordonnancer, afin de déterminer quelle tâche peut être exécutée à un instant donné, quelle quantité de processeur elle peut utiliser et pour combien de temps.

Nous nous intéresserons dans un premier temps aux ordonnanceurs des SEG les plus utilisés avant d'étudier ceux des principaux hyperviseurs déployés en production.

Windows

Windows fait appel à un tourniquet et aux priorités pour ordonnancer les processus [win13].

Il sélectionne toujours le processus avec la plus haute priorité ; si, au cours de l'exécution d'un processus, un autre processus avec une priorité plus élevée se trouve prêt à être exécuté, le premier processus sera préempté au bénéfice du second.

Tous les processus ayant la même priorité sont traités sur un pied d'égalité ; chaque processus est autorisé à s'exécuter pendant une durée fixe, après quoi il est préempté pour laisser la place au processus suivant, à la manière d'un tourniquet.

Linux

L'ordonnanceur par défaut de Linux, le *Completely Fair Scheduler* (CFS) [cfs13], recourt à un indicateur, le temps d'exécution virtuel (TEV), pour ordonnancer les processus conventionnels. Le TEV d'un processus est égal au temps d'utilisation du processeur, pondéré par la priorité de ce processus et par le nombre de processus

qui sont prêts à être exécutés. CFS sélectionne toujours le processus qui a le plus petit TEV, jusqu'à trouver un autre processus avec un TEV inférieur.

Chaque processeur dispose de son propre lot de processus à exécuter ; CFS fait en sorte que le nombre de processus en attente d'exécution soit à peu près le même pour chaque processeur, afin d'équilibrer la charge de travail [sch13].

CFS prend en compte l'affinité d'un processus avec un processeur donné. Ceci permet de favoriser l'utilisation du cache de chaque processeur, augmentant ainsi les performances des processus.

KVM

KVM délègue le travail d'ordonnancement à l'ordonnanceur de Linux, CFS, étant donné que les machines virtuelles KVM apparaissent comme des processus standards sous Linux [KKL+07].

Xen

Le concept de temps d'exécution virtuel (TEV) utilisé par CFS est similaire à celui employé par l'un des ordonnanceurs historiques de Xen, le *Borrowed Virtual Time scheduler* (BVT) [Chi07]. BVT calcule le TEV de chaque machine virtuelle en comptabilisant son temps d'exécution, qu'il pondère par le poids associé à la machine virtuelle. BVT déduit éventuellement un facteur de diminution du TEV afin d'obtenir le temps d'exécution virtuel effectif (TEVE) ; ce facteur de diminution est fourni par une machine virtuelle souhaitant être temporairement ordonnancée en priorité, étant donné que BVT sélectionne toujours la machine virtuelle avec le plus faible TEVE.

BVT a été éclipsé par un nouvel ordonnanceur, le *Credit Scheduler* (CS) [Chi07], qui assigne à chaque machine virtuelle :

- un poids, qui détermine quelle proportion de temps processeur la machine virtuelle peut utiliser ;
- une capacité, qui représente la quantité maximale de temps processeur utilisable par la machine virtuelle.

CS alloue périodiquement des crédits aux machines virtuelles en fonction de leur poids. CS ordonnance ensuite les machines virtuelles suivant un tourniquet. Lorsqu'une machine virtuelle a épuisé ses crédits ou atteint sa capacité, elle laisse la place à une autre machine. De cette manière, les machines virtuelles avec un poids faible tendent à consommer leurs crédits rapidement, libérant ainsi le processeur pour celles qui ont un poids plus élevé.

ESX

ESX tient compte, pour chaque machine virtuelle, du ratio entre (i) son temps d'utilisation effectif du processeur et (ii) sa part, c'est-à-dire le temps pendant lequel cette machine virtuelle était autorisée à utiliser le processeur [VMw09]. La machine virtuelle ayant le ratio le plus faible est exécutée en priorité.

Un utilisateur d'ESX a plusieurs moyens d'influer sur la part allouée à une machine virtuelle, en lui assignant :

- un poids ; une machine virtuelle d'un poids deux fois supérieur à celui d'une autre machine peut utiliser deux fois plus longtemps le processeur ;

– une réservation de capacité processeur ; l'ordonnanceur garantit qu'une machine virtuelle ayant une réservation peut accéder au minimum à la capacité réservée ;
– une limite d'utilisation du processeur ; une machine virtuelle ne peut pas utiliser plus de temps processeur que la limite qui lui a été fixée.

ESX veille à équilibrer la charge de travail entre les processeurs, en déplaçant des machines virtuelles d'un processeur à un autre ; il déplace en priorité les machines virtuelles qui font peu appel aux processeurs, afin de ne pas pénaliser celles qui sont susceptibles d'avoir stocké des données dans les caches des processeurs.

Hyper-V

Hyper-V met en œuvre des concepts similaires à ceux présents dans ESX, à savoir : la définition d'un poids, d'une réservation ou d'une limite pour influer sur l'utilisation du processeur par une machine virtuelle [hyp13].

Bilan

Nous avons trouvé des similitudes au cours de l'étude des ordonnanceurs de processus et de machines virtuelles, notamment concernant les notions de préemption et de tourniquet, de priorité d'un processus ou de poids assigné à une machine virtuelle, de temps d'exécution virtuel, d'affinité pour un processeur, et d'équilibrage de charge entre les processeurs.

En fait les besoins en matière d'ordonnancement sont si proches qu'un hyperviseur hébergé a tout intérêt à déléguer le travail d'ordonnancement à son SEG hôte, comme KVM le fait avec Linux.

4.1.3 Gestion de la mémoire

Après le processeur, la mémoire est la seconde ressource à laquelle un gestionnaire doit porter une attention particulière.

Allocation

Quand une tâche est créée, le gestionnaire doit lui allouer de la mémoire. Comme pour l'ordonnancement, un SEG et un hyperviseur natif disposent de leur propre allocateur, tandis qu'un hyperviseur hébergé se repose sur l'allocateur de son SEG hôte.

Pagination

Un processus ne travaille pas directement avec les adresses de mémoire physique, autrement il pourrait entrer en conflit avec un autre processus ; à la place, il utilise des adresses de mémoire virtuelle.

La mémoire virtuelle est communément mise en œuvre via le mécanisme de pagination [Tan01] ; le SEG doit alors maintenir pour chaque processus une table des pages qui contient la correspondance entre les adresses physiques d'une part et les adresses virtuelles d'autre part.

L'utilisation de la pagination est un peu plus compliquée avec un hyperviseur. Comme dans le cas d'un processus, un SEG invité non modifié ne doit pas avoir

accès aux adresses physiques, sous peine de conflit avec un autre SEG invité. Un SEG invité s'occupe alors de la traduction des adresses virtuelles de l'invité en adresses physiques de l'invité. L'hyperviseur est quant à lui responsable :

- soit de la traduction des adresses virtuelles de l'invité en adresses physiques de l'hôte, dans le cas de la *pagination fantôme* (ou *shadow paging*) [SN05], qui est une technique d'émulation ;
- soit de la traduction des adresses physiques de l'invité en adresses physiques de l'hôte, dans le cas de la *pagination imbriquée* (ou *nested paging*) [Adv08], ce qui nécessite des extensions matérielles de virtualisation sur l'architecture x86.

Partage de mémoire

La pagination peut être utilisée pour partager des pages de mémoire physique entre plusieurs applications afin de réduire la consommation mémoire.

Le partage de mémoire est notamment utilisé par le mécanisme de *copie sur écriture* (ou *copy-on-write* [Tan01]) : lorsqu'un processus fils est créé à partir d'un processus père, le premier partage initialement ses pages mémoire avec le second ; les pages mémoire sont protégées en écriture ; lorsqu'un processus tente de changer le contenu d'une page mémoire, le SEG lui une copie qu'il peut modifier à loisir.

Les processus partagent également les pages mémoire contenant le code des bibliothèques logicielles.

Les processus peuvent partager plus de mémoire si le SEG permet de faire de la déduplication, comme c'est le cas de Linux grâce au programme KSM (*Kernel Shared Memory*) [AEW09]. KSM compare les pages mémoire des processus ; si des pages ont un contenu identique, KSM les fusionne ; le mécanisme de copie sur écriture est alors utilisé pour permettre à un processus essayant d'écrire dans une page d'obtenir une copie modifiable. Notons que la déduplication mémoire a d'abord été mise en œuvre dans les hyperviseurs (cf. chapitre 2).

Le gain résultant de la déduplication mémoire augmente sensiblement dans le contexte de la virtualisation : un nœud peut héberger plusieurs machines virtuelles qui utilisent le même SEG invité, ce qui consomme une quantité importante de mémoire inutilement ; il est alors intéressant que l'hyperviseur puisse fusionner les pages mémoire identiques appartenant à des machines virtuelles différentes [Wal02]. Cette fonctionnalité est présente au sein de l'hyperviseur dans le cas d'un type I, alors qu'elle peut être localisée dans le SEG hôte dans le cas d'un type II.

Fichier d'échange

Lorsque la mémoire vient à manquer, même après avoir fait appel à la déduplication, il faut alors envisager de transférer le contenu de certaines pages dans le *fichier d'échange* [Tan01], qui est stocké sur le disque dur.

Un SEG cherche à transférer en priorité le contenu des pages les moins utilisées afin de préserver au maximum les performances des programmes impliqués.

Un hyperviseur n'a malheureusement pas connaissance de la fréquence d'utilisation des pages. Il pourrait transférer le contenu de pages choisies aléatoirement, mais cela ne serait pas efficace. Heureusement, l'hyperviseur peut dialoguer avec les SEG invités grâce à la technique de *ballooning* [Wal02] (présentée au chapitre 2) pour savoir quelles sont les pages les moins utilisées.

Bilan

En résumé, nous avons vu que les SEG et les hyperviseurs proposaient des fonctionnalités équivalentes pour la gestion de la mémoire, notamment en matière d'allocation, de pagination, de partage de mémoire (et en particulier de déduplication mémoire) et de recours au fichier d'échange.

4.1.4 Synthèse

En conclusion de cette section, nous avons montré que les SEG et les hyperviseurs proposaient un grand nombre de fonctionnalités similaires pour la gestion des ressources et des tâches au niveau local.

Nous avons observé que beaucoup de ces fonctionnalités, qui avaient été mises en place à l'origine dans les SEG, ont été réutilisées par les hyperviseurs ; l'inverse est également vrai, puisque la déduplication mémoire a été popularisée par les hyperviseurs avant d'être mise en œuvre dans un SEG.

4.2 Comparaison dans un contexte distribué

Après la gestion des ressources et des tâches dans un contexte local, nous nous intéresserons à leur gestion dans un contexte distribué.

Nous nous attacherons à comparer les fonctionnalités proposées par les systèmes d'exploitation distribués (SED) d'une part et les logiciels de gestion d'infrastructure virtualisée (LGIV) d'autre part.

4.2.1 Cycle de vie des tâches

La gestion des tâches inclue leur déploiement sur l'infrastructure, leur migration d'un nœud à un autre, et la prise d'instantanés.

Déploiement des tâches

Les SED sont susceptibles de déployer des processus d'au moins deux manières différentes : (i) par duplication de processus à distance [GHS02] ou (ii) par création de processus à distance [LSNG03]. Dupliquer un processus implique de transférer l'état du processus père sur le nœud où le processus fils doit être créé. À l'inverse, créer un processus à distance signifie qu'un processus est créé sur le nœud de destination sans cloner un processus existant.

Ces fonctionnalités sont similaires à celles qui sont mises en œuvre par certains LGIV. Ainsi, le projet Snowflock [CWm+09] permet de faire de la duplication de machine virtuelle à distance : une machine virtuelle peut être clonée en plusieurs exemplaires qui fonctionnent sur des nœuds différents ; peu d'informations sont initialement copiées ; un clone ne récupère le contenu des pages mémoire que lors du premier accès, depuis une image mémoire immuable de la machine virtuelle d'origine. Cependant, les LGIV utilisent plus fréquemment la création distante de machine virtuelle ; ils doivent alors mettre à disposition l'image disque de la machine virtuelle sur le nœud de destination, avant de démarrer ladite machine virtuelle.

Migration des tâches

Une fois qu'une tâche est déployée sur un nœud, le gestionnaire peut décider de la migrer sur un autre nœud lorsqu'il estime que c'est nécessaire.

Migration de processus La migration de processus [MDP+00] peut être utilisée pour faire de l'équilibrage de charge dynamique, pour réaliser des opérations de maintenance, ou pour améliorer l'efficacité des accès aux données (en rapprochant un processus des données sur lesquelles il travaille).

Il y a deux prérequis pour pouvoir migrer un processus :

- le processus doit avoir un identifiant unique, afin d'être identifiable où qu'il soit dans l'infrastructure ;
- les différentes informations relatives au processus (valeurs des registres, espace d'adressage et descripteurs des fichiers ouverts) doivent être extraites.

Une fois que les informations d'un processus ont été extraites, plusieurs types d'algorithmes peuvent être utilisés pour migrer le contenu des pages mémoire.

Après avoir migré, un processus fonctionne normalement sur le nœud de destination ; cependant, certains appels système ne peuvent pas être effectués directement car les ressources ne sont plus toujours disponibles localement. C'est le problème des *dépendances résiduelles*. Il existe plusieurs manières de le résoudre :

- en envoyant une requête au nœud qui est capable de traiter l'appel système ;
- en envoyant une requête à un système distribué qui va traiter l'appel système ;
- en demandant une copie de la ressource liée à l'appel système afin de pouvoir y accéder localement.

En résumé, la migration de processus est une fonctionnalité importante proposée par les SED, mais elle est difficile à mettre en place.

Migration de machine virtuelle La migration (à chaud) de machine virtuelle [CFH+05] est plus facile à mettre en œuvre que la migration de processus car il n'y a pas de dépendance résiduelle sur le nœud d'origine. Par conséquent, la plupart des hyperviseurs permettent de migrer des machines virtuelles.

Des recherches récentes visent à réduire la durée d'inactivité d'une machine virtuelle qui est inhérente à la migration en recourant à des algorithmes dérivés de ceux utilisés pour la migration de processus [HNIm11].

Prise d'instantanés

La prise d'instantanés [CL85] a un fonctionnement similaire à celui de la migration, dans la mesure où il faut extraire l'état du processus (respectivement de la machine virtuelle), sauf que cet état est stocké sur le disque dur au lieu d'être transféré sur un autre nœud via le réseau.

La prise d'instantanés est plus facile à mettre en place pour les machines virtuelles que pour les processus, pour les mêmes raisons que celles évoquées pour la migration.

Bilan

En résumé, les LGIV se sont inspirés des SED pour proposer des mécanismes de gestion des machines virtuelles, comme le déploiement, la migration et la prise d'instantanés. Comme souligné précédemment, les mécanismes de migration et de prise d'instantanés sont plus faciles à mettre en place pour les machines virtuelles que pour les processus.

Étant donné que de fortes similarités existent déjà entre les SED et les LGIV à ce niveau, la littérature sur les SED ne semble pas pouvoir aider à améliorer encore la gestion des machines virtuelles dans un contexte distribué.

4.2.2 Ordonnancement

Les mécanismes de déploiement, de migration et de prise d'instantanés sont susceptibles d'être mis à profit par l'ordonnanceur global de l'infrastructure, notamment s'il ordonnance les tâches de manière dynamique.

Ordonnancement dynamique et systèmes d'exploitation distribués

L'ordonnancement dynamique est utilisé par les SED (comme OpenMosix, OpenSSI et Kerrighed [LGV+05]), principalement pour équilibrer la charge de travail en migrant des processus depuis les nœuds très chargés vers les nœuds peu chargés.

Une migration peut être initiée par le nœud qui envoie le processus, celui qui le reçoit, ou bien les deux [Tan01].

Dans le cas d'une politique à l'initiative de l'envoyeur, un nœud très chargé (l'envoyeur) recherche un nœud peu chargé pour se délester d'une partie de ses processus.

À l'inverse, lorsqu'une politique à l'initiative du destinataire est utilisée, un nœud peu chargé (le destinataire) recherche un nœud plus chargé pour le soulager d'une partie de sa charge de travail.

Enfin, dans une politique symétrique, un nœud peut demander à recevoir de nouveaux processus ou au contraire à se débarrasser d'une partie d'entre eux, en fonction de sa charge.

Ordonnancement dynamique et logiciels de gestion d'infrastructure virtualisée

Contrairement aux SED, la plupart des LGIV qui font de l'ordonnancement dynamique (comme Entropy [HLM+09, HDL11], vSphere [VMw11] et SCVMM [Mic12]) adoptent une approche centralisée dans laquelle un nœud de service unique est responsable de l'ordonnancement pour toute l'infrastructure.

Plusieurs propositions ont été faites afin de décentraliser l'ordonnancement des machines virtuelles [YMF+10, MMP11, RC11, FRM12], mais les prototypes qui en sont issus continuent de recourir à des points centralisés, notamment pour la collecte des informations sur l'état des ressources de l'infrastructure. Nous reviendrons plus en détail sur ces propositions dans le chapitre suivant.

À l'heure actuelle, comme nous l'avons déjà remarqué au chapitre 3, l'équilibrage de charge et la consolidation sont les deux politiques d'ordonnancement dynamique les plus fréquemment mises en œuvre au sein des LGIV.

Bilan

En résumé, pour ce qui touche à l'ordonnancement dynamique des tâches dans une infrastructure distribuée, les LGIV pourraient être améliorés grâce aux recherches sur les SED ; en effet, les premiers ont tendance à recourir à une approche centralisée, tandis que les seconds ont depuis longtemps adopté une architecture décentralisée.

4.2.3 Gestion de la mémoire

Nous venons d'étudier comment les mécanismes d'ordonnancement pouvaient être étendus dans un contexte distribué ; nous allons maintenant faire de même avec la gestion de la mémoire.

Deux solutions ont été présentées dans la littérature concernant la gestion globale de la mémoire : l'ordonnancement global et les mémoires partagées distribuées.

Via l'ordonnancement

Nous avons montré dans la section 4.1.3 que la déduplication permettait de réduire la quantité de mémoire utilisée sur un nœud ; cette fonctionnalité a un équivalent distribué pour les machines virtuelles, qui a été mis en œuvre dans le projet sur les *Memory Buddies* [WLS+09].

L'idée principale est de faire en sorte que chaque SEG invité (i) analyse le contenu mémoire de la machine virtuelle qui l'héberge et (ii) envoie cette information à un nœud de service.

Le nœud de service analyse ensuite les similarités de contenu mémoire entre les machines virtuelles et effectue des migrations afin de regrouper les machines virtuelles qui présentent le plus de similitudes.

Les hyperviseurs prennent alors le relais et utilisent la déduplication pour réduire la consommation mémoire sur chacun des nœuds concernés.

À notre connaissance, les SED ne proposent pas de fonctionnalité équivalente à celle mise en œuvre dans les *Memory Buddies*.

Via une mémoire partagée distribuée

Le recours à une mémoire partagée distribuée est une autre façon de gérer globalement la mémoire de l'infrastructure ; elle fournit « l'abstraction de la mémoire partagée sur des nœuds dont la mémoire est physiquement distribuée » [Esk96].

Ce sujet de recherche très populaire a été appliqué aux SED, et plus particulièrement à ceux visant à créer des systèmes à image unique (comme OpenMosix, OpenSSI et Kerrighed) ; en revanche, cette abstraction est peu utilisée par les LGIV. Elle a néanmoins été mise en œuvre dans vNUMA [CH09] et ScaleMP vSMP Foundation [vsm13], deux hyperviseurs qui agrègent les ressources processeur et mémoire de plusieurs nœuds homogènes. Remarquons au passage que cette approche s'applique sur des infrastructures de taille réduite ; vNUMA a ainsi été évalué sur 8 nœuds, tandis que ScaleMP certifie que son hyperviseur peut agréger jusqu'à 128 nœuds.

Bilan

La recherche sur les LGIV semble être en avance sur celle relative aux SED en matière de gestion de la mémoire dans un contexte distribué, puisqu'elle a porté son attention non seulement sur l'abstraction de mémoire partagée distribuée, mais également sur l'extension de la déduplication mémoire.

Cependant, la plupart des LGIV ne permettent pas de faire de déduplication mémoire en distribué. Par ailleurs, le prototype existant suit une approche centralisée, ce qui limite sa capacité à passer à l'échelle.

4.2.4 Synthèse

En résumé, les LGIV proposent des fonctionnalités similaires aux SED pour la gestion des tâches et des ressources dans un contexte distribué.

Cependant, les premiers peuvent être améliorés (i) en généralisant les mécanismes de gestion de la mémoire et (ii) en décentralisant l'ordonnancement.

4.3 Conclusion

Dans ce chapitre, nous avons comparé les fonctionnalités de gestion proposées par les logiciels de gestion d'infrastructure virtualisée (LGIV) et les systèmes d'exploitation distribués (SED), à l'échelle d'un nœud isolé et au niveau de l'infrastructure tout entière. Nous nous sommes plus particulièrement concentrés sur la gestion du cycle de vie et sur l'ordonnancement des tâches.

Nous avons dans un premier temps développé les réflexions initiées par des travaux précédents concernant les systèmes d'exploitation et les hyperviseurs [HWF+05, HUL06, REH07]. Ceci nous a permis de souligner (i) les similarités entre ces deux catégories de gestionnaires et (ii) le fait que les avancées dans l'un des champs de recherche pouvaient bénéficier à l'autre.

Nous avons ensuite étendu notre étude au contexte distribué, ce qui nous a permis d'identifier des axes de contribution, notamment en matière de décentralisation de l'ordonnancement dynamique des machines virtuelles.

Nous approfondirons ce dernier point dans les chapitres suivants. Nous reviendrons tout d'abord sur les solutions existantes avant de présenter notre propre approche, qui constitue la contribution majeure de ces travaux de thèse.

5

Ordonnancement dynamique des machines virtuelles

La comparaison des logiciels de gestion d'infrastructure virtualisée et des systèmes d'exploitation distribués nous a permis d'identifier des axes de contribution pour les premiers, notamment en matière de décentralisation de l'ordonnancement dynamique des machines virtuelles.

Dans ce chapitre, nous revenons sur les principaux choix de conception qui ont une incidence sur l'architecture des ordonnanceurs ; nous détaillons également les limites d'une approche centralisée ; enfin, nous étudions les solutions proposées dans la littérature pour décentraliser l'ordonnancement des machines virtuelles, en distinguant les approches hiérarchiques des approches multi-agents.

Sommaire

5.1 Architecture des ordonnanceurs

L'ordonnancement dynamique des machines virtuelles permet de prendre en compte les fluctuations de leur charge de travail afin d'optimiser leur placement. Les principales étapes du processus d'ordonnancement dynamique sont au nombre de trois. La première est la collecte d'informations, au cours de laquelle l'ordonnanceur récupère des données sur les ressources des nœuds de calcul et l'utilisation

que les machines virtuelles en font. La deuxième étape est la prise de décision : l'ordonnanceur calcule un plan de reconfiguration à partir des informations récupérées, c'est-à-dire une série d'opérations (par exemple des migrations) à effectuer sur des machines virtuelles. Enfin, le plan de reconfiguration est appliqué au cours de la dernière étape.

Plusieurs solutions existent pour mener à bien ces différentes étapes. Nous nous intéresserons plus particulièrement à la collecte d'informations et à la prise de décision. En effet, l'application du plan de reconfiguration consiste uniquement à envoyer des ordres de migration aux différents hyperviseurs.

5.1.1 Collecte d'informations

Plusieurs choix doivent être faits concernant la collecte d'informations sur l'état des ressources de l'infrastructure [Rot94].

Il est tout d'abord nécessaire de déterminer quels nœuds sont en charge de la collecte de ces informations : un seul nœud de service dédié, dans le cadre d'une architecture centralisée ; plusieurs nœuds de service organisés de manière hiérarchique ; ou encore tous les nœuds de calcul, dans le cas d'une approche multi-agents.

Il faut ensuite définir quels nœuds de calcul sont impliqués lors de chaque collecte d'informations : tous, quelques-uns, ou encore un nombre variable.

Il faut également établir si les nœuds de calcul sont volontaires ou non dans le processus de collecte d'informations, autrement dit s'ils choisissent quand envoyer des informations au collecteur, ou bien s'ils se contentent de répondre aux sollicitations de ce dernier.

Dans le cas où les nœuds sont volontaires, il faut spécifier à quel moment ils doivent envoyer les informations au collecteur : périodiquement, ou bien seulement lorsque certaines conditions sur l'état de leurs ressources sont réunies.

5.1.2 Prise de décision

De la même manière que pour le processus de collecte d'informations, la phase de prise de décision nécessite d'effectuer plusieurs choix [Rot94].

Il faut notamment préciser quels nœuds sont responsables de la prise de décision. Là encore, il est possible de recourir à une approche centralisée, hiérarchique ou multi-agents.

Dans le cas d'une approche multi-agents, il faut spécifier quels nœuds prennent les décisions, afin d'éviter les actions contradictoires. Cela peut nécessiter de mettre en place un mécanisme de synchronisation.

5.2 Limites de l'approche centralisée

Comme nous l'avons montré aux chapitres 3 et 4, les gestionnaires d'infrastructure virtualisée qui permettent d'ordonnancer dynamiquement les machines virtuelles tendent à recourir à une approche centralisée et périodique (cf. figure 5.1).

Chaque étape de l'ordonnancement dure un certain temps, les étapes de calcul et d'application d'un plan de reconfiguration étant les plus longues. La durée de chacune de ces étapes est d'autant plus significative que le nombre de nœuds et de machines virtuelles à gérer est grand. Or, de nos jours, les infrastructures des plus

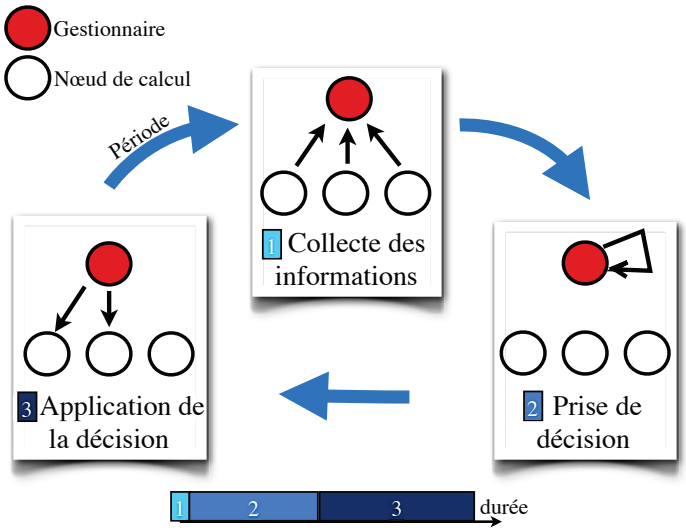


FIGURE 5.1 – Ordonnancement dynamique centralisé et périodique

grands organismes atteignent des tailles conséquentes ; ainsi, deux acteurs importants de l'informatique en nuage, Rackspace et OVH, ont annoncé qu'ils disposaient respectivement d'environ 80 000 et 120 000 nœuds [who13].

Les ordonnanceurs centralisés en charge de grandes infrastructures ne sont pas aptes à réagir rapidement lorsque certains nœuds se trouvent surutilisés, ce qui risque de conduire à une violation des niveaux de qualité de service fixés en accord avec les clients.

Par ailleurs, plus le temps de reconfiguration est long et plus il y a de risques que la charge de travail des machines virtuelles fluctue et rende le plan de reconfiguration caduc avant même qu'il soit entièrement appliqué.

Enfin, le nœud hébergeant l'ordonnanceur représente un point unique de défaillance : s'il tombe, ou s'il n'est plus connecté aux autres nœuds au travers du réseau, il n'y a plus d'ordonnancement possible.

Une façon de traiter les problèmes évoqués jusqu'à présent consiste à décentraliser l'ordonnancement. Nous aborderons par la suite les différentes solutions qui ont été proposées dans la littérature.

5.3 Présentation d'une approche hiérarchique : Snooze

Une première manière de décentraliser l'ordonnancement est de recourir à une approche hiérarchique.

Une telle approche constitue un compromis entre l'approche centralisée et les approches multi-agents qui sont plus décentralisées.

Afin d'en illustrer une mise en œuvre, nous nous attarderons sur la proposition Snooze.

5.3.1 Présentation

Snooze [FM12, FRM12] repose sur une architecture hiérarchique à trois niveaux, qui est constituée :

– des *contrôleurs locaux* (CL) ;

– des *gestionnaires de groupe* (GG) ;
– du *chef de groupe* (CG).

Chaque nœud héberge un CL. Les CL sont répartis à leur démarrage entre plusieurs GG (chaque GG tourne sur son propre nœud de service) ; l'affectation d'un CL n'est remise en cause qu'en cas de défaillance de son GG ; chaque CL envoie périodiquement à son GG des informations concernant l'état des ressources de son nœud et les machines virtuelles qu'il héberge. Les GG sont tous placés sous les ordres du CG (le CG étant élu au sein des GG, il délègue la gestion de ses CL aux autres GG) ; chaque GG envoie périodiquement au CG les informations agrégées remontées par les CL dont il a la charge (cf. figure 5.2).

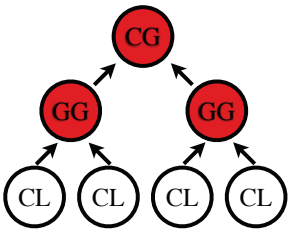


FIGURE 5.2 – Snooze – collecte d'informations

Snooze prend en compte les ressources processeur, mémoire et réseau lors du processus d'ordonnancement.

Plusieurs facteurs peuvent déclencher un ordonnancement :
– la soumission d'une nouvelle machine virtuelle sur l'infrastructure ; le CG choisit le GG qui va placer la machine virtuelle, et ce dernier choisit sur quel nœud démarrer cette machine virtuelle ;
– l'optimisation périodique d'un critère défini par l'administrateur (par exemple pour faire de la consolidation) ; chaque GG cherche à optimiser périodiquement ce critère au sein des nœuds qui sont à sa charge ;
– la surutilisation ou la sous-utilisation d'un nœud ; le CL concerné avertit son GG, qui corrige le problème.

Le prototype a été évalué en conditions réelles sur le banc de test Grid'5000, au travers d'une expérience qui a impliqué 500 machines virtuelles réparties sur 136 nœuds de calcul.

5.3.2 Discussion

Snooze présente l'avantage de répartir le travail d'ordonnancement entre plusieurs entités logicielles, les gestionnaires de groupe, tout en assurant la continuité de l'ordonnancement en cas de défaillance d'un des éléments du système. Cependant, l'architecture demeure partiellement centralisée, puisque le chef de groupe a une vision globale (bien que simplifiée) de toute l'infrastructure et qu'il reçoit toutes les soumissions de machines virtuelles, ce qui peut limiter le passage à l'échelle.

Par ailleurs, l'évaluation du prototype se concentre sur la soumission de machines virtuelles, sans étudier l'optimisation périodique ou encore la résolution des problèmes de surutilisation ou de sous-utilisation d'un nœud.

Enfin, la répartition des CL entre les GG soulève plusieurs difficultés : si un GG est en charge de nombreux CL, l'ordonnancement peut prendre du temps et ne pas être réactif ; à l'inverse, si un GG a peu de CL sous ses ordres, cela diminue

d'autant les possibilités de replacement des machines virtuelles, ce qui implique en particulier que le GG ne sera pas nécessairement capable de corriger les problèmes de surutilisation des nœuds. Une amélioration envisagée (mais pas encore mise en œuvre à notre connaissance) consisterait à faire coopérer les GG entre eux lorsque l'un des GG ne parvient pas à trouver de solution.

5.4 Présentation des approches multi-agents

D'autres propositions ont été élaborées afin de décentraliser encore plus le travail d'ordonnancement ; ces approches sont qualifiées d'approches multi-agents, car elles recourent à l'installation d'un agent logiciel sur chacun des nœuds de l'infrastructure à gérer, le travail d'ordonnancement étant réparti entre les agents.

5.4.1 *A Bio-Inspired Algorithm for Energy Optimization in a Self-organizing Data Center* [BDNDM10]

Présentation

Cette approche vise à optimiser la consommation électrique de l'infrastructure. L'algorithme d'ordonnancement s'apparente à de la consolidation.

Dans cette approche, chaque agent peut envoyer un ou plusieurs scouts en reconnaissance. Un scout est une entité logicielle qui voyage sur les nœuds voisins de son nœud d'origine afin de récolter des informations sur les ressources disponibles. Une fois de retour sur son nœud d'origine, un scout partage ces informations avec l'agent local. L'agent décide alors s'il est judicieux de délester son nœud d'une partie de ses machines virtuelles ; ceci a d'autant plus de chance de se produire si le nœud est peu chargé (et inversement). En cas de migration, l'agent choisit comme nœud de destination celui qui consomme le moins d'électricité et dont les ressources (processeur et mémoire) sont les plus utilisées. Le nœud d'origine est éteint lorsqu'il n'héberge plus aucune machine virtuelle.

L'approche a été validée au travers de simulations portant au plus sur 1 000 nœuds de calcul (le nombre de machines virtuelles n'a pas été indiqué dans l'article).

Discussion

Cette approche est originale, dans la mesure où elle est à notre connaissance la première à décentraliser complètement l'ordonnancement dynamique des machines virtuelles : chaque agent a une connaissance partielle du système et s'occupe d'ordonnancer uniquement les machines virtuelles hébergées sur son nœud. En outre, cette approche est tolérante aux pannes : si un scout disparaît suite à la défaillance d'un nœud, il suffit que son nœud d'origine crée un nouveau scout pour que l'ordonnancement se poursuive.

Ses concepteurs remarquent néanmoins que l'approche occasionne un grand nombre de migrations. Ils soulignent également la nécessité de borner le nombre de nœuds dont un scout peut se rappeler afin de transférer moins de données sur le réseau. Ensuite, les résultats expérimentaux montrent que le taux de consolidation est assez éloigné de l'optimal. Par ailleurs, les informations récoltées par les scouts peuvent être périmées quand ils reviennent sur leur nœud d'origine, et cela pour

deux raisons : des machines virtuelles peuvent avoir été déplacées entre-temps, et la consommation en ressources des machines virtuelles est susceptible d'avoir fluctué. En outre, aucun mécanisme de synchronisation n'a *a priori* été mis en place afin d'empêcher plusieurs nœuds d'envoyer simultanément leurs machines virtuelles sur un même nœud de destination, ce qui peut créer des situations de surcharge. Enfin, cette approche ne prend pas en compte les dépendances séquentielles, c'est-à-dire le fait que la migration d'une machine virtuelle peut nécessiter de déplacer une autre machine virtuelle au préalable pour pouvoir être réalisée.

5.4.2 *Dynamic Resource Allocation in Computing Clouds through Distributed Multiple Criteria Decision Analysis* [YMF+10]

Présentation

L'algorithme d'ordonnancement mis en œuvre par cette approche est la consolidation.

Dans cette approche, un calcul d'ordonnancement n'est déclenché que lorsqu'un agent constate que les ressources de son nœud sont surutilisées ou sous-utilisées. Dans le premier cas, il cherche à se débarrasser d'une machine virtuelle qui permettrait de repasser sous le seuil de surcharge ; dans le second cas, il tente d'expulser toutes les machines virtuelles de son nœud. Pour chaque machine virtuelle à migrer, l'agent contacte un médiateur (qui est en charge de la collecte des informations sur l'état des ressources des nœuds) ; le médiateur renvoie à l'agent la liste des nœuds ayant suffisamment de ressources disponibles pour accueillir la machine virtuelle à migrer ; l'agent choisit alors le nœud avec le moins de ressources (processeur, mémoire et réseau) disponibles.

Cette approche a été validée au travers de simulations ayant comporté jusqu'à 20 000 machines virtuelles et 2 500 nœuds de calcul.

Discussion

Cette approche met en œuvre un mécanisme de synchronisation afin d'éviter que plusieurs nœuds tentent d'envoyer simultanément leurs machines virtuelles sur un même nœud de destination. De plus, les simulations effectuées par les concepteurs de cette approche montrent que celle-ci permet de réduire grandement le nombre de migrations de machines virtuelles par rapport à une approche heuristique centralisée ; cet avantage est cependant obtenu en sacrifiant le taux de consolidation.

Par ailleurs, si cette approche est bien décentralisée au niveau de la prise de décision, elle demeure centralisée en matière de collecte des informations, avec les limites que cela implique en termes de passage à l'échelle et de tolérance aux pannes (si le médiateur tombe, les agents ne peuvent plus migrer leurs machines virtuelles).

Enfin, cette approche ne gère pas les dépendances séquentielles pour les migrations des machines virtuelles.

5.4.3 *Server Consolidation in Clouds through Gossiping* [MBP11]

Présentation

La politique d'ordonnancement mise en œuvre dans cette approche est la consolidation.

Chaque agent communique de manière périodique avec chacun de ses voisins (le voisinage change périodiquement et aléatoirement). À chaque communication, le nœud qui a le moins de machines virtuelles envoie à l'autre nœud autant de machines virtuelles que celui-ci peut en recevoir.

Cette approche a été évaluée au travers de simulations avec 40 000 machines virtuelles et 10 000 nœuds de calcul.

Discussion

Cette approche est entièrement décentralisée. Elle est également tolérante aux pannes, dans la mesure où, si un nœud tombe, les autres nœuds peuvent continuer à s'échanger des machines virtuelles. En outre, elle permet d'approcher le taux de consolidation optimal.

En revanche, elle ne tient pas compte de l'hétérogénéité des machines virtuelles, et en particulier de leur consommation en ressources, qui influe sur le coût de migration. De plus, cette approche ne permet pas de gérer les situations où les ressources des nœuds sont surutilisées. Par ailleurs, elle entraîne un grand nombre de migrations [FME12]. Enfin, cette approche ne gère pas les dépendances séquentielles pour les migrations des machines virtuelles.

5.4.4 *Self-economy in Cloud Data Centers – Statistical Assignment and Migration of Virtual Machines* [MMP11]

Présentation

Cette approche permet de faire de la consolidation.

La soumission d'une nouvelle machine virtuelle sur l'infrastructure se fait au travers d'un nœud de service. Ce nœud interroge tous les agents pour savoir où placer la machine virtuelle. Chaque agent évalue alors une fonction de probabilité pour déterminer s'il est judicieux que son nœud héberge la nouvelle machine virtuelle ; si c'est le cas, l'agent en avertit le nœud de service. Le nœud de service choisit ensuite aléatoirement l'un des nœuds de calcul qui a répondu pour placer la machine virtuelle.

Par ailleurs, chaque agent évalue régulièrement une autre fonction de probabilité pour déterminer s'il aurait intérêt à libérer son nœud d'une ou plusieurs machines virtuelles. La probabilité de migrer une machine virtuelle augmente lorsque le taux d'utilisation du processeur du nœud (i) dépasse un seuil de surutilisation, ou au contraire (ii) passe en-dessous d'un seuil de sous-utilisation. L'agent choisit alors une machine virtuelle à migrer et contacte le nœud de service pour savoir où la placer, comme s'il s'agissait d'une soumission d'une nouvelle machine virtuelle.

Cette approche a été évaluée au travers de simulations ayant impliqué au plus 2 400 machines virtuelles et 100 nœuds de calcul.

Discussion

Cette approche permet de décentraliser la collecte des informations sur les nœuds de calcul ; en revanche, le choix du nœud de destination d'une machine virtuelle est effectué de manière centralisée. Ceci présente l'avantage d'éviter que plusieurs nœuds envoient en même temps leurs machines virtuelles à un même nœud de destination. En revanche, cela peut également poser problème, pour des raisons de

tolérance aux pannes (si le nœud de service est défaillant) ou de passage à l'échelle (le nœud de service est en effet très sollicité lorsque le taux d'utilisation de l'infrastructure s'approche du maximum, étant donné que le nombre de migrations augmente de manière exponentielle, ainsi que le montrent les résultats des simulations). Enfin, cette approche ne gère pas les dépendances séquentielles pour les migrations de machines virtuelles.

5.4.5 *A Distributed and Collaborative Dynamic Load Balancer for Virtual Machine* [RC11]

Présentation

Cette approche a été conçue pour faire de l'équilibrage de charge.

Un ordonnancement est déclenché (i) lors de l'arrivée d'une nouvelle machine virtuelle sur l'infrastructure, (ii) lorsque les ressources (processeur et mémoire) d'un nœud sont surutilisées ou bien encore (iii) lorsqu'un nœud est mis en maintenance. La tâche d'ordonnancement est alors ajoutée à une file d'attente sur le nœud où elle a été créée. Cette tâche d'ordonnancement peut être traitée par l'agent local, ou bien par l'agent d'un autre nœud, via un mécanisme de vol de tâches ; nous supposerons par la suite que la tâche d'ordonnancement est traitée en local, pour simplifier les explications. Dans le cas d'un événement lié à la surutilisation des ressources d'un nœud, l'agent ne sait pas dès le départ quelle machine virtuelle migrer ; il commence donc par récupérer des informations sur les machines virtuelles locales, avant de sélectionner la machine virtuelle qui consomme le moins de ressources pour la migrer. Une fois que l'agent sait quelle machine virtuelle migrer, il doit lui trouver une destination ; il collecte dans un premier temps des informations sur l'ensemble des nœuds de l'infrastructure, avant de sélectionner le nœud qui a le plus de ressources disponibles. L'agent migre alors la machine virtuelle sur le nœud de destination.

Cette approche a été évaluée au travers de simulations qui ont impliqué jusqu'à 1 000 machines virtuelles et 140 nœuds de calcul.

Discussion

Cette approche présente comme avantages (i) de bien répartir le travail d'ordonnancement entre les nœuds et (ii) d'être tolérante aux pannes (si un nœud est défaillant, cela n'empêchera pas le reste des nœuds de poursuivre l'ordonnancement des machines virtuelles restantes).

En revanche, pour chaque tâche d'ordonnancement, la collecte d'informations et la prise de décision nécessitent de prendre en compte tous les nœuds de l'infrastructure, ce qui limite la capacité de cette approche à passer à l'échelle.

De plus, la décision n'est pas forcément applicable car l'état des ressources du nœud de destination peut avoir changé (une vérification est effectuée avant que les machines virtuelles soient placées/migrées) ; cela implique de relancer un nouveau calcul d'ordonnancement pour trouver une solution.

Enfin, cette approche ne gère pas les dépendances séquentielles pour les migrations de machines virtuelles.

5.4.6 *A Case for Fully Decentralized Dynamic VM Consolidation in Clouds* [FME12]

Présentation

Cette approche améliore la proposition présentée dans [MBP11] pour faire de la consolidation.

Là encore, chaque nœud n'a connaissance que des nœuds de son voisinage, voisinage qui change périodiquement et aléatoirement. Cependant, les nœuds ne communiquent pas deux à deux pour s'échanger des machines virtuelles. En revanche, chaque agent tente périodiquement de réserver tous les nœuds de son voisinage ; il échoue si l'un des nœuds est déjà réservé par un autre agent, et il passe alors son tour ; à l'inverse, s'il réussit, il devient responsable de l'ordonnancement des machines virtuelles de son voisinage. Pour cela, il récupère les informations sur la consommation en ressources (processeur, mémoire et réseau) des machines virtuelles ; la façon de replacer ces machines virtuelles dépend quant à elle de l'algorithme de consolidation utilisé. Une fois l'ordonnancement terminé, l'agent libère les nœuds pour qu'ils puissent être réservés par un autre agent.

Cette approche a été évaluée avec différents algorithmes de consolidation (dont [MBP11]). Cette évaluation a été réalisée au travers de simulations qui ont impliqué jusqu'à 6 048 machines virtuelles et 1 008 nœuds de calcul.

Discussion

Comme [MBP11], cette approche est décentralisée et partiellement tolérante aux pannes. En effet, la défaillance d'un nœud quelconque ne met pas un point d'arrêt à l'ordonnancement ; en revanche, la défaillance du nœud responsable de l'ordonnancement sur un voisinage met un terme à l'ordonnancement sur ce voisinage ; de plus, les nœuds étant réservés pour l'ordonnancement, ils ne pourront pas être réservés pour un autre ordonnancement une fois qu'ils auront été répartis dans d'autres voisinages, ce qui fera échouer toute tentative d'ordonnancement sur ces voisinages. Ce problème met en lumière une autre faiblesse de cette approche : il faudrait un algorithme plus sophistiqué pour élire un meneur au sein de chaque voisinage, afin de garantir une élection rapide et sans risque d'inter-blocages.

Contrairement à [MBP11], cette approche tient compte de la consommation réelle en ressources des machines virtuelles et permet de faire du surbooking de ressources. Cependant, cette approche ne permet pas de traiter efficacement les situations de surcharge, étant donné que l'ordonnancement est réalisé de manière périodique sur un voisinage de taille fixe : la période d'ordonnancement est de 30 secondes, et s'il n'est pas possible de corriger le problème avec les nœuds du voisinage, il faudra attendre que ce voisinage ait changé pour tenter à nouveau de trouver une solution.

Enfin, là encore, cette approche ne gère pas les dépendances séquentielles pour les migrations de machines virtuelles.

5.5 Conclusion

Dans ce chapitre, nous avons étudié différentes approches permettant de faire de l'ordonnancement dynamique de machines virtuelles (cf. tableaux 5.1 et 5.2), en

nous concentrant sur la tolérance aux pannes et le passage à l'échelle (autrement dit la gestion efficace d'infrastructures comportant un grand nombre de nœuds et de machines virtuelles).

Nous sommes tout d'abord revenus sur les principaux choix de conception qui ont une incidence sur l'architecture des ordonnanceurs, notamment au niveau de la collecte d'informations sur l'infrastructure et de la prise de décision.

Nous avons ensuite présenté les limites de l'approche centralisée, qui ne permet pas de corriger rapidement les violations de qualité de service, au point qu'un plan de reconfiguration peut devenir caduc avant d'avoir été complètement appliqué.

Nous avons ensuite détaillé un exemple d'architecture hiérarchique au travers de Snooze [FRM12, FM12], qui permet de répartir le travail d'ordonnancement entre plusieurs entités logicielles, qui est tolérant aux pannes mais qui est toujours partiellement centralisé.

Nous nous sommes enfin intéressés aux approches multi-agents, qui permettent de répartir tout ou partie du travail d'ordonnancement entre tous les nœuds de l'infrastructure [BDNDM10, YMF+10, MBP11, MMP11, RC11, FME12]. Nous avons cependant relevé plusieurs limitations. Concernant le passage à l'échelle, certaines approches nécessitent d'avoir des informations sur toute l'infrastructure pour prendre une décision [YMF+10, RC11], et/ou recourent à un nœud de service centralisé [YMF+10, MMP11]. Ce nœud de service n'est pas tolérant aux pannes, et sa défaillance met un point d'arrêt à l'ordonnancement des machines virtuelles sur l'infrastructure. L'ordonnancement entraîne parfois de nombreuses migrations de machines virtuelles [BDNDM10, MBP11], sans nécessairement permettre d'optimiser le critère d'ordonnancement choisi [BDNDM10]. Par ailleurs, aucune des approches ne permet de gérer les dépendances séquentielles au niveau des migrations. Enfin, [FME12] ne permet pas de corriger rapidement les problèmes de surutilisation des ressources d'un nœud, dans le cas d'une infrastructure surbookée.

De notre point de vue, une approche décentralisée idéale pour ordonnancer dynamiquement les machines virtuelles doit :

- pouvoir gérer des infrastructures surbookées ;
- se montrer extrêmement réactive pour corriger les problèmes de surutilisation des ressources d'un nœud ;
- proposer des solutions d'une qualité équivalente à celles obtenues avec une approche centralisée ;
- limiter le nombre de migrations à effectuer ;
- gérer les dépendances séquentielles lors de l'application d'un plan de reconfiguration ;
- disposer d'un mécanisme de synchronisation efficace pour éviter les conflits lors de l'application simultanée de plusieurs plans de reconfiguration ;
- fonctionner correctement lorsque des nœuds sont ajoutés ou retirés de l'infrastructure.

C'est avec ces objectifs en tête que nous avons conçu la proposition DVMS (*Distributed Virtual Machine Scheduler*), que nous présenterons au chapitre suivant.

Nom	Déclenchement d'un ordonnancement	Politiques d'ordonnancement	Ressources discriminantes pour l'ordonnancement
[BDNDM10]	retour d'un scout sur son nœud d'origine	optimisation énergétique (≈ consolidation)	*processeur *mémoire *consommation électrique
[YMF+10]	*création d'une VM *surutilisation d'un nœud *sous-utilisation d'un nœud	consolidation	*processeur *mémoire *bande passante réseau
[MBP11]	période	consolidation	nombre de VMs hébergées
[MMP11]	*création d'une VM *surutilisation d'un nœud *sous-utilisation d'un nœud	consolidation	processeur
[RC11]	*création d'une VM *surutilisation d'un nœud *opération de maintenance sur un nœud	équilibrage de charge	*processeur *mémoire
Snooze	*création d'une VM *surutilisation d'un nœud *sous-utilisation d'un nœud *période	*création d'une VM : tourniquet *surutilisation, sous-utilisation, période : non évalués	*processeur *mémoire *réseau
[FME12]	période (30 secondes)	consolidation	*processeur *mémoire *réseau

TABLEAU 5.1 – Synthèse des propositions académiques en matière d'ordonnancement dynamique décentralisé de machines virtuelles (1/2)

Nom	Choix d'une machine virtuelle (VM) à migrer	Choix d'un nœud de destination	Type et taille de la plus grande expérience
[BDNDM10]	1)récupération des informations sur les VMs hébergées localement 2)choix des VMs migrables qui ont intérêt à migrer sur un des nœud traversés par un scout (remarque : la probabilité de migrer des VMs est inversement proportionnelle à la charge du nœud qui les héberge)	1)récupération des informations sur les nœuds apportées par les scouts 2)sélection du nœud qui consomme le moins d'électricité, et qui a les ressources les plus utilisées	*simulation *1 000 nœuds
[YMF+10]	1)récupération des informations sur les VMs hébergées localement 2a)surutilisation : choix de la VM permettant de repasser sous le seuil de surcharge 2b)sous-utilisation : toutes les VMs hébergées localement	1)récupération auprès du médiateur de tous les nœuds pouvant héberger la VM 2)choix du nœud avec le moins de ressources disponibles	*simulation *20 000 VMs / 2 500 nœuds
[MBP11]	1)communication avec un des 20 nœuds du voisinage (reconstruit aléatoirement toutes les x secondes) 2)le nœud ayant le moins de VMs envoie autant de VMs que l'autre nœud peut en recevoir	cf. choix d'une VM à migrer	*simulation *40 000 VMs / 10 000 nœuds
[MMP11]	1)récupération des informations sur les VMs hébergées localement 2)choix de la VM ?	1)le médiateur demande à tous les nœuds s'ils peuvent accueillir la VM 2)seuls répondent les nœuds qui peuvent et qui ont peu de ressource processeur disponible 3)le médiateur choisit un nœud au hasard parmi ceux-ci	*simulation *2 400 VMs / 100 nœuds
[RC11]	1)récupération des informations sur les VMs hébergées localement 2)choix de la VM ayant les plus faibles besoins en ressources	1)récupération des informations sur tous les nœuds 2)choix du nœud avec le plus de ressources disponibles	*simulation *1 000 VMs / 140 nœuds
Snooze	1)les contrôleurs locaux envoient périodiquement (10 s) les informations sur leurs VMs à leur gestionnaire de groupe 2)le gestionnaire de groupe en déduit quelles VMs migrer	1)les contrôleurs locaux envoient périodiquement (10 s) les informations sur leur nœud à leur gestionnaire de groupe 2)le gestionnaire de groupe en déduit sur quels nœuds migrer les VMs	*expérience sur système réel *500 VMs / 136 nœuds
[FME12]	1)récupération des informations sur les VMs hébergées par les 16 nœuds du voisinage (reconstruit aléatoirement toutes les 10 s) 2)le choix de la VM dépend de l'algorithme utilisé ; ACO tend à favoriser les VMs ayant les besoins en ressources les plus importants	1)récupération des informations sur les 16 nœuds du voisinage 2)le choix du nœud dépend de l'algorithme utilisé ; ACO tend à favoriser les nœuds avec peu de ressources disponibles	*simulation *6 048 VMs / 1 008 nœuds

TABLEAU 5.2 – Synthèse des propositions académiques en matière d'ordonnancement dynamique décentralisé de machines virtuelles (2/2)

III

DVMS : une solution coopérative et décentralisée pour l'ordonnancement dynamique des machines virtuelles

6

DVMS : une proposition pour l'ordonnancement coopératif et réactif des machines virtuelles

Dans le chapitre précédent, nous avons étudié les différentes contributions concernant l'ordonnancement autonome, dynamique et décentralisé des machines virtuelles.

Bien que des progrès significatifs aient été réalisés, nous avons constaté le manque de solution permettant à la fois de (i) gérer des infrastructures surbookées, (ii) corriger rapidement les problèmes de surcharge, (iii) proposer des solutions d'une qualité similaire à celles obtenues avec une approche centralisée, (iv) limiter le nombre de migrations, (v) gérer les dépendances séquentielles, (vi) prévenir les conflits dus à l'application simultanée de plusieurs plans de reconfiguration, et (vii) prendre en compte l'ajout ou le retrait de nœuds.

C'est avec ces principes en tête que nous avons conçu une nouvelle approche : DVMS (*Distributed Virtual Machine Scheduler*). DVMS est déployé sous la forme d'un réseau d'agents organisés en anneau et qui coopèrent entre eux afin de traiter au plus vite les événements (liés à la surutilisation ou à la sous-utilisation des ressources d'un nœud) qui se produisent sur l'infrastructure ; DVMS peut traiter plusieurs événements de manière simultanée et indépendante en partitionnant dynamiquement l'infrastructure, chaque partition ayant une taille appropriée à l'événement qui lui est associé.

Dans ce chapitre, nous détaillons les idées fondatrices de DVMS et nous présentons leur mise en œuvre dans un prototype. Les éléments exposés dans ce chapitre ont fait l'objet de deux publications internationales, l'une dans un atelier de travail [QL12] et l'autre dans une revue [QLS12].

Sommaire

6.1 Fondements de DVMS

6.1.1 Hypothèses de départ

Nous travaillons avec les hypothèses suivantes concernant l'infrastructure à gérer :

1. les nœuds peuvent être identiques ou non ;
2. chaque nœud peut communiquer avec n'importe quel autre nœud à travers le réseau ;
3. chaque nœud héberge un hyperviseur, hyperviseur qui est le même pour tous les nœuds ;
4. les machines virtuelles sont hétérogènes ; ce sont des boîtes noires, dont nous ne connaissons que la consommation en ressources processeur et mémoire ;
5. chaque machine virtuelle peut être migrée sur n'importe quel nœud, ce qui implique que :
   – l'image disque de chaque machine virtuelle est accessible par n'importe quel nœud, que ce soit grâce à un espace de stockage partagé ou bien en migrant tout ou partie de l'image disque lorsque cela est nécessaire ;
   – le réseau dispose d'une bande passante suffisante pour migrer une machine virtuelle rapidement (en l'espace de quelques secondes) ;
   – dans le cas où les nœuds n'ont pas tous le même modèle de processeur, l'hyperviseur utilisé permet quand même d'effectuer des migrations (c'est notamment le cas pour KVM).

D'un point de vue logiciel, les nœuds sont répartis sur un anneau ; cette structure garantit que, si un message émis par un nœud a besoin de traverser tous les nœuds de l'infrastructure, il ne passera qu'une seule fois par chaque nœud avant de revenir à son envoyeur.

Ce message prend concrètement la forme d'un événement ; un ordonnancement étant déclenché lorsqu'un événement se produit sur l'infrastructure, ceci assure un traitement rapide de l'événement. Nous nous intéresserons plus particulièrement aux événements liés à la surutilisation et à la sous-utilisation des ressources d'un nœud ; ces événements sont définis par des seuils d'utilisation fixés par l'administrateur.

Chaque événement est associé à une partition.

Définition 6.1 (Partition)
Une partition est constituée de l'ensemble des nœuds réservés pour la résolution d'un événement donné.

Le partitionnement de l'infrastructure est nécessaire car il permet d'éviter les conflits entre ordonnanceurs lors de l'application des plans de reconfiguration. Supposons un instant que deux ordonnanceurs, O_1 et O_2, puissent manipuler les mêmes machines virtuelles, et que O_1 calcule son plan de reconfiguration plus rapidement que O_2 ; l'application du plan de reconfiguration de O_1 peut très bien invalider le plan de O_2, si jamais O_1 (i) migre des machines virtuelles que O_2 aurait également voulu déplacer ou si O_1 (ii) utilise des ressources qui auraient été nécessaires pour que O_2 puisse réaliser certaines migrations ; O_2 aurait alors passé du temps à calculer un plan de reconfiguration pour rien, vu que ce plan se révélerait inutilisable, ce que nous voulons éviter à tout prix.

Chaque partition possède deux nœuds spéciaux : l'initiateur et le meneur.

Définition 6.2 (Initiateur)
L'initiateur d'une partition est le nœud sur lequel s'est initialement produit l'événement associé à la partition.

Définition 6.3 (Meneur)
Le meneur d'une partition est le nœud qui mène le calcul d'ordonnancement visant à résoudre l'événement associé à la partition ; le meneur de la partition est susceptible de changer au cours du traitement de l'événement.

6.1.2 Présentation de la procédure de traitement d'un événement

La procédure de traitement d'un événement [QL12] est initiée lorsqu'un nœud N_i constate qu'il y a un problème, par exemple lorsque ses ressources sont surutilisées (cf. figures 6.1 et 6.2) ; il génère alors un événement et s'auto-réserve pour la résolution de cet événement (cf. figure 6.2(a)). Il transfère ensuite l'événement à son voisin, le nœud N_{i+1}, c'est-à-dire le nœud qui est situé juste après lui sur l'anneau (en parcourant ce dernier dans le sens horaire).

Si N_{i+1} est déjà impliqué dans une autre partition, il transfère directement l'événement au nœud N_{i+2} ; dans le cas contraire, N_{i+1} intègre la nouvelle partition (cf. figure 6.2(b)) et vérifie que l'événement est toujours valide. Si l'événement n'est plus valide (par exemple parce que la consommation en ressources des machines virtuelles hébergées par l'initiateur a fluctué), N_{i+1} annule les réservations afin de détruire la partition et de permettre ainsi à tous ses nœuds de prendre part à la résolution d'autres événements. À l'inverse, si l'événement est toujours valide, N_{i+1} avertit tous les nœuds de la partition qu'il est le nouveau meneur ; en retour, il reçoit des informations concernant (i) les ressources de chaque nœud et (ii) les ressources consommées par les machines virtuelles hébergées par chacun de ces nœuds. Il lance alors un calcul d'ordonnancement ; si aucune solution n'est trouvée, l'événement est alors transmis au nœud N_{i+2}.

N_{i+2} répète les mêmes opérations, à savoir : auto-réservation (s'il est libre, cf. figure 6.2(c)), vérification de la validité de l'événement, avertissement des nœuds de la partition concernant le changement de meneur, récupération des informations sur les ressources et les machines virtuelles de chacun de ces nœuds et lancement d'un calcul d'ordonnancement. Si une solution est trouvée, N_{i+2} applique le plan de reconfiguration (ce qui permet de résoudre l'événement) puis annule les réservations, afin de détruire la partition et de permettre aux nœuds de prendre part à la résolution d'autres événements.

Notons que, si N_{i+2} n'avait pas trouvé de solution, la partition aurait continué à s'agrandir jusqu'à ce qu'une solution soit trouvée ou bien que l'événement ait fait le tour de l'anneau. Dans ce dernier cas, le problème aurait été considéré comme insoluble et la partition aurait été détruite ; ce comportement n'est pas idéal et nous proposerons une manière de l'améliorer par la suite.

L'agrandissement progressif de la partition vise à adapter sa taille en fonction de la complexité du problème à résoudre ; cela permet de prendre en compte le moins de nœuds possible, d'où un temps d'ordonnancement réduit qui permet de traiter le problème au plus vite.

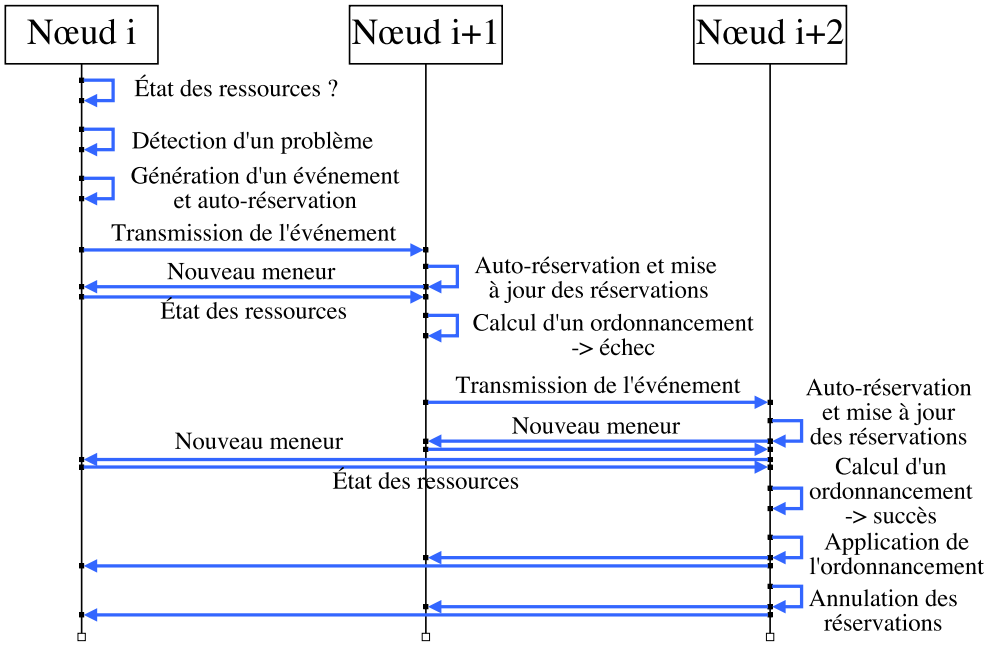


FIGURE 6.1 – Détail de la procédure de traitement d'un événement

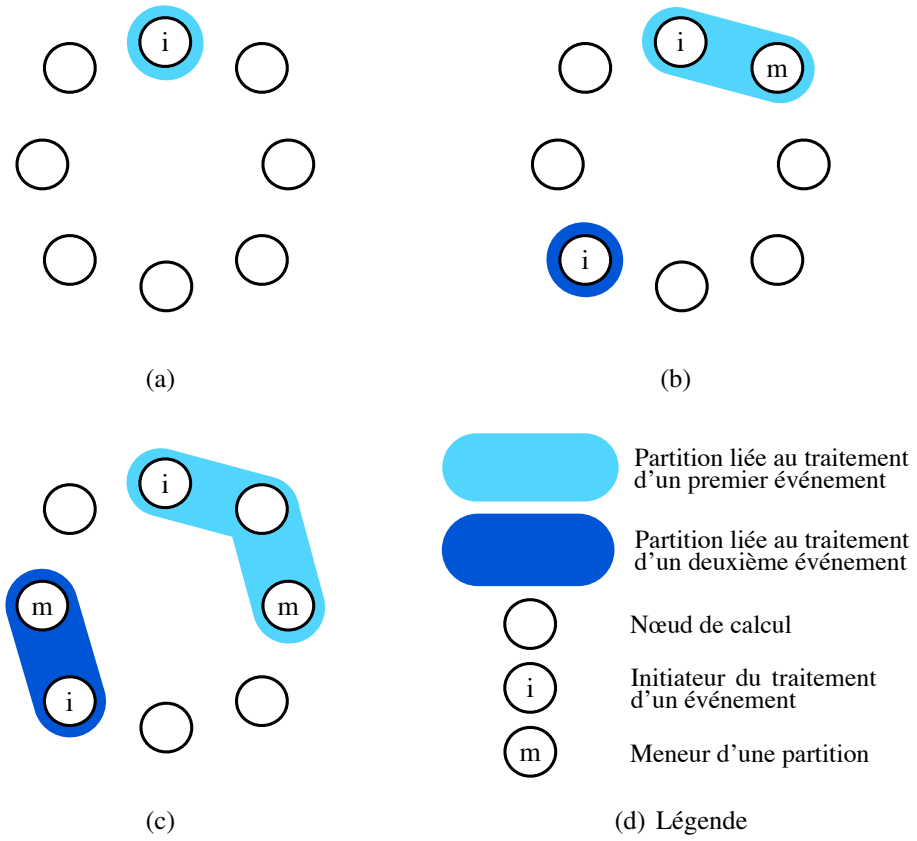


FIGURE 6.2 – Parallélisation du traitement de deux événements

Précisons que si nous faisons en sorte que le calcul d'un plan de reconfiguration puisse prendre en compte plus de deux nœuds à la fois, c'est pour offrir plus de possibilités de migration à un ordonnanceur gérant les dépendances transitives : il peut en effet être nécessaire de déplacer une ou plusieurs machine(s) virtuelle(s) avant de pouvoir en migrer une autre qui permettrait de résoudre le problème.

Soulignons également le fait que, à chaque changement de meneur, ce dernier récupère des informations sur l'état des ressources de chaque nœud de la partition avant de lancer un calcul d'ordonnancement ; ceci est nécessaire car la consommation en ressources des machines virtuelles peut fluctuer tout au long de l'existence de la partition.

Remarquons enfin que plusieurs événements peuvent être traités en parallèle et de manière indépendante, comme le montrent les figures 6.2(b) et 6.2(c).

6.1.3 Accélération du parcours de l'infrastructure

Par la suite, nous avons étudié plusieurs possibilités pour améliorer le traitement des événements ; la première amélioration que nous avons mise en œuvre a consisté à définir des *raccourcis* pour accélérer le parcours de l'anneau [QLS12].

Définition 6.4 (Raccourci)
Un raccourci est un chemin vers le premier nœud situé en dehors d'une partition.

Prenons l'exemple (cf. figure 6.3(a)) d'une partition bleu clair P_1 qui a besoin de s'agrandir, mais qui se heurte à une partition bleu foncé P_2 qui est constituée d'un nombre relativement important de nœuds contigus. L'algorithme présenté précédemment nécessite de transférer de nœud en nœud l'événement E_1 associé à P_1, jusqu'à arriver au premier nœud libre en dehors de la partition bleu foncé. Cependant, la situation est différente si chaque nœud de P_2 possède un raccourci vers le premier nœud situé en dehors de cette partition (cf. figure 6.3(b)). Ainsi, lorsque E_1 est transféré à l'initiateur de P_2, ce dernier est en mesure de faire suivre immédiatement l'événement au premier nœud en dehors de sa partition.

Chaque nœud doit vérifier que son raccourci pointe toujours sur le premier nœud en dehors de sa partition ; cette vérification est effectuée à chaque agrandissement de ladite partition. À cette occasion, le nouveau meneur communique son raccourci à ses compagnons de partition ; ce raccourci pointe sur le voisin du meneur (ou sur le nœud pointé par le raccourci du voisin, si ce dernier fait partie de la même partition que le meneur) ; un nœud ne met à jour son raccourci que si celui-ci désignait jusqu'à présent le nœud qui est devenu le nouveau meneur. Cette mise à jour sélective permet de gérer les situations où la partition n'est pas constituée de nœuds contigus (cf. figure 6.3(c)) ; ainsi, les deux premiers nœuds de la partition P_1 conservent leur raccourci vers l'initiateur de la partition P_2, tandis que le raccourci du meneur de P_1 référence le dernier nœud libre de l'infrastructure.

6.1.4 Assurance de trouver une solution si elle existe

Une autre amélioration de la procédure de traitement d'un événement est de garantir qu'une solution sera toujours trouvée si elle existe [QLS12].

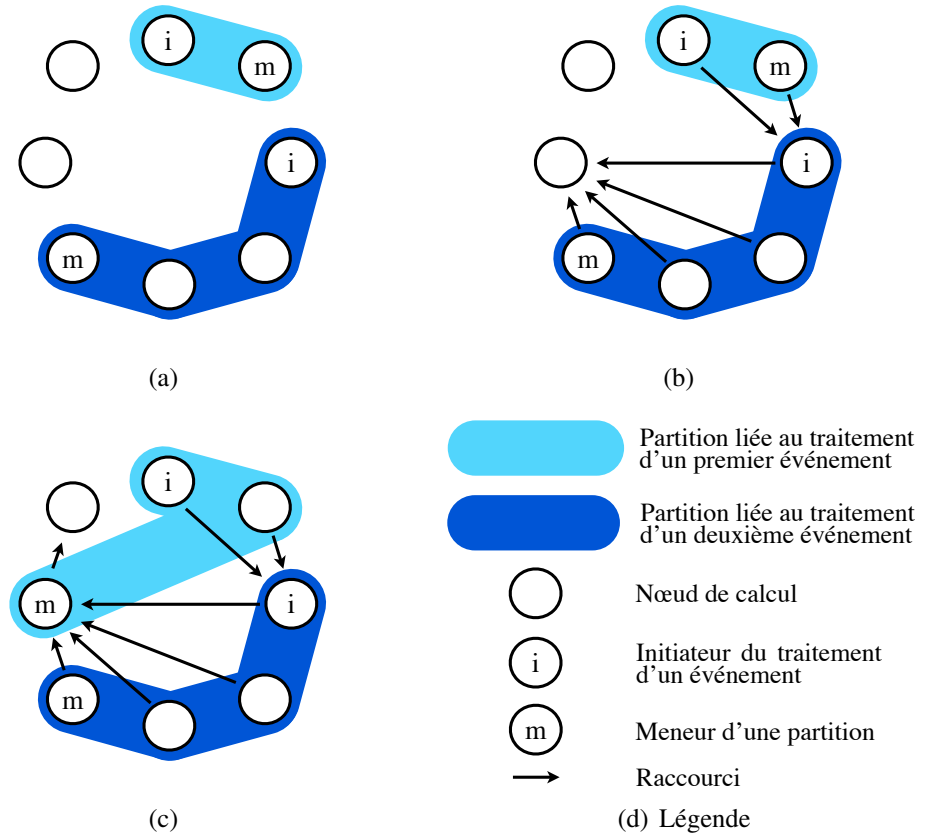


FIGURE 6.3 – Utilisation de raccourcis pour agrandir plus rapidement une partition

Prérequis

Un premier pas en ce sens consiste à cesser de détruire une partition lorsqu'un événement donné revient à son initiateur sans avoir été résolu ; il est en effet possible que des nœuds, préalablement impliqués dans d'autres partitions, se soient libérés entre-temps et puissent être utilisés pour résoudre l'événement qui nous intéresse.

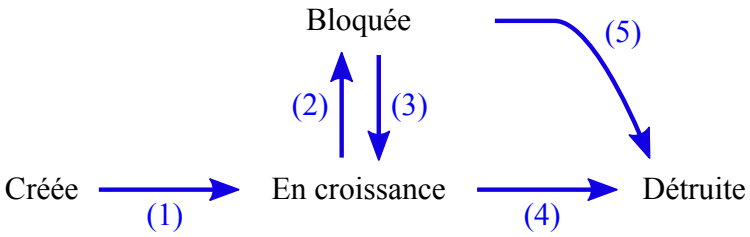
Cette étape n'est cependant pas suffisante, car elle peut créer des situations de pénurie de nœuds libres.

Définition 6.5 (Pénurie)
Une pénurie se produit lorsqu'au moins deux partitions ont besoin de s'agrandir alors qu'il ne reste plus de nœuds libres dans l'infrastructure.

Vue d'ensemble de la gestion des pénuries

Afin d'échapper aux pénuries, notre approche consiste à fusionner les partitions deux à deux. Fusionner deux partitions ne garantit pas que les événements associés pourront être résolus ; cependant, cela offre plus de possibilités de migrations à l'ordonnanceur. Cela est d'autant plus vrai lorsque deux événements complémentaires sont impliqués, par exemple deux événements liés respectivement à la surutilisation et à la sous-utilisation des ressources d'un nœud.

Afin d'identifier les partitions qu'il pourrait être intéressant de fusionner, nous leur associons des états : créée, en croissance, bloquée ou détruite (cf. figure 6.4).



(1) Un événement se produit, son traitement débute
(2) L'événement fait le tour de l'anneau sans être résolu
(3) Un ou plusieurs nœud(s) intègre(nt) la partition
(4) L'événement est résolu
(5) Aucune solution n'existe, l'événement est ignoré

FIGURE 6.4 – États associés à une partition

Lorsqu'une partition est créée, son état initial est spécifié comme étant *en croissance*. Elle conserve cet état tant que l'événement qui lui est associé n'est pas revenu à son initiateur ; lorsque cela se produit, la partition est alors identifiée comme étant *bloquée*. Cela ne signifie pas nécessairement qu'il n'y a plus de nœuds libres dans l'infrastructure, mais c'est une possibilité.

L'événement continue de parcourir l'anneau tant qu'il n'a pas été résolu et qu'il reste au moins un nœud en dehors de la partition P_1 qui lui est associée ; afin d'accélérer ce parcours, nous utilisons les raccourcis présentés précédemment. L'arrivée de l'événement sur un nœud externe à P_1 soulève trois possibilités :

1. le nœud est membre d'une partition P_2 en croissance ; l'événement est alors transféré au premier nœud externe à P_2 ; il ne serait en effet pas judicieux de fusionner P_1 avec P_2, cette dernière étant susceptible de résoudre son événement toute seule, elle pourrait être pénalisée en devant absorber plus de nœuds que nécessaire ;
2. le nœud est membre d'une partition P_2 qui est elle aussi bloquée ; les deux partitions tentent alors de fusionner ; si cette tentative échoue, l'événement est transféré au premier nœud externe à P_2 ; si elle réussit, la partition résultante repasse en croissance ; nous reviendrons plus en détails par la suite sur le fonctionnement de la fusion des partitions ;
3. le nœud est libre ; il est alors intégré dans la partition P_1, qui repasse en croissance.

Si une solution est trouvée, la procédure de traitement se termine avec succès ; si au contraire tous les nœuds de l'infrastructure ont été absorbés par la partition sans qu'une solution ait été trouvée, le problème est garanti comme étant insoluble.

Les différentes étapes de l'algorithme de gestion des situations de pénurie de nœuds sont synthétisées sur la figure 6.5.

Détail du processus de fusion de deux partitions

Comme promis, nous revenons maintenant sur le processus de fusion de deux partitions.

Tout d'abord, nous faisons une différence entre la partition demandeuse et la partition exécutrice.

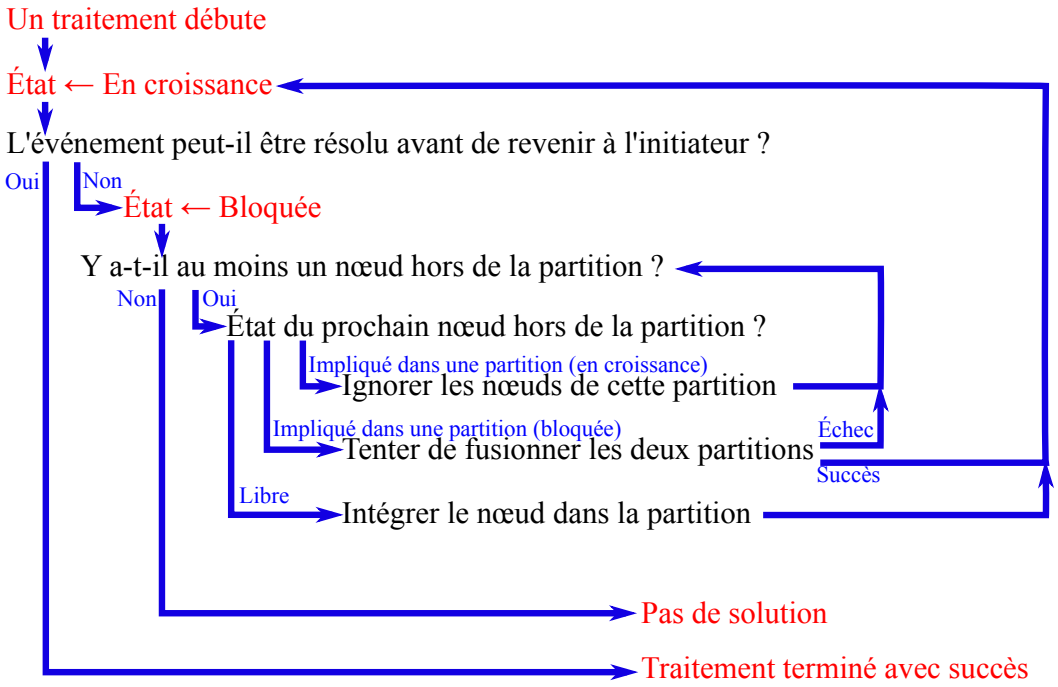


FIGURE 6.5 – Algorithme de gestion des pénuries de nœuds

Définition 6.6 (Partition demandeuse)
La partition demandeuse est la partition qui effectue la demande de fusion.

Définition 6.7 (Partition exécutrice)
La partition exécutrice est la partition qui réalise la fusion.

Afin d'éviter que les deux partitions ne cherchent à endosser simultanément le rôle de demandeuse, nous définissons un ordre total en associant un identifiant unique à chaque nœud, en fonction de sa place sur l'anneau ; nous avons choisi arbitrairement que la partition avec le meneur d'identifiant le plus faible serait la partition demandeuse.

Il faut tout d'abord vérifier qu'aucune des deux partitions n'est déjà impliquée dans un processus de fusion ; le meneur de la partition exécutrice se charge alors de la fusion des partitions et des événements associés :

- le type et l'initiateur de l'événement fusionné sont les mêmes que ceux de l'événement de la partition demandeuse ;
- le meneur de la partition fusionnée est le même que celui de la partition exécutrice ;
- la partition fusionnée est constituée des nœuds des deux partitions ;
- les raccourcis sont mis à jour pour éviter qu'ils pointent sur des nœuds de la partition fusionnée (ce qui peut arriver si des raccourcis de la partition demandeuse pointaient sur des nœuds de la partition exécutrice, et inversement).

Une fois les partitions fusionnées, un calcul d'ordonnancement est lancé afin de résoudre l'événement fusionné.

Prenons l'exemple de trois partitions (cf. figure 6.6) pour résumer les notions évoquées jusqu'à présent en matière de gestion des situations de pénurie de nœuds. Les deux partitions bleu clair sont initialement bloquées (cf. figure 6.6(a)) ; l'identifiant du meneur de la partition du bas (6) est inférieur à celui du meneur de la

partition du haut (8) ; la partition du bas endosse donc le rôle de demandeuse, et celle du haut celui d'exécutrice ; la partition fusionnée a la même couleur que celle de l'ex-partition du bas, pour rappeler le type de l'événement (cf. figure 6.6(b)). Si les deux partitions restantes sont également bloquées, le processus de fusion recommence ; cette fois, la partition de droite est la demandeuse et celle de gauche l'exécutrice ; la partition résultante a la même couleur que l'ex-partition de droite, là encore pour rappeler le type de l'événement (cf. figure 6.6(c)). Si un ordonnancement sur la partition fusionnée ne permet pas de résoudre l'événement associé, nous avons alors la garantie qu'aucune solution n'existait, puisque tous les nœuds de l'infrastructure ont été pris en compte.

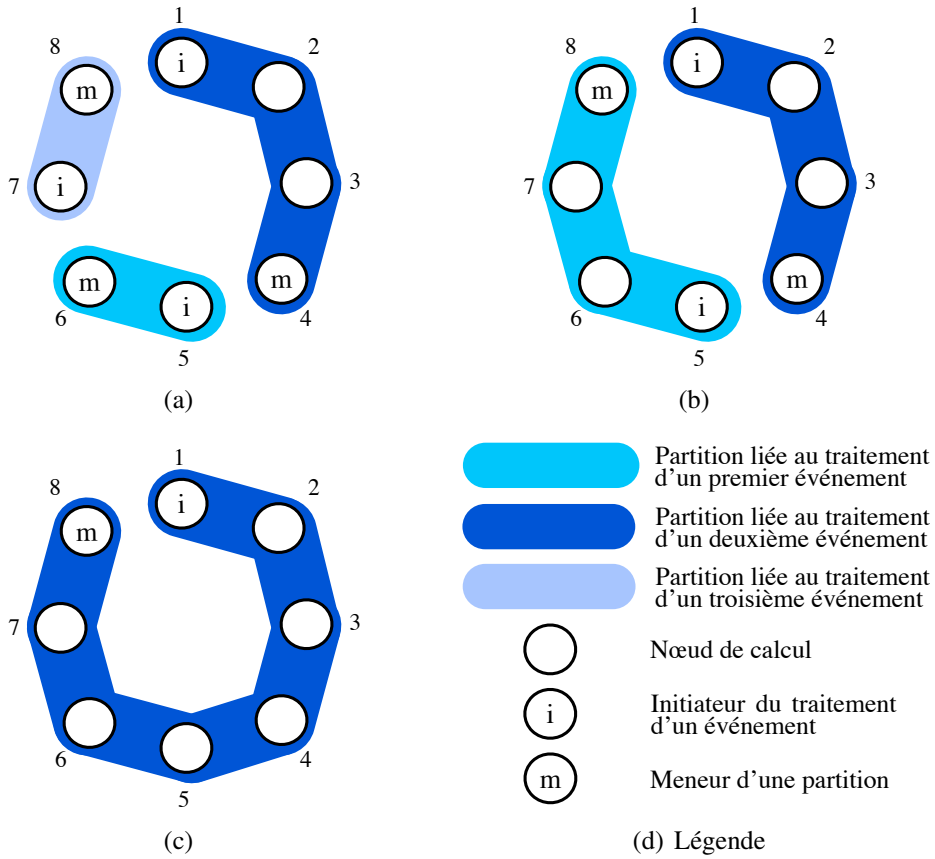


FIGURE 6.6 – Exemples de fusion de partitions visant à pallier le manque de nœuds libres dans l'infrastructure

6.2 Mise en œuvre

Les concepts présentés jusqu'à présent ont été mis en œuvre dans un prototype écrit en Java.

Le prototype permet à l'heure actuelle de gérer des événements liés à la surutilisation et à la sous-utilisation des ressources d'un nœud.

6.2.1 Architecture d'un agent

Le prototype est déployé sur une infrastructure sous la forme d'un ensemble d'agents logiciels, chaque agent étant chargé de la gestion d'un nœud ; un agent DVMS est constitué de cinq composants principaux : une base de connaissance, un observateur, un client, un serveur et un ordonnanceur (cf. figure 6.7).

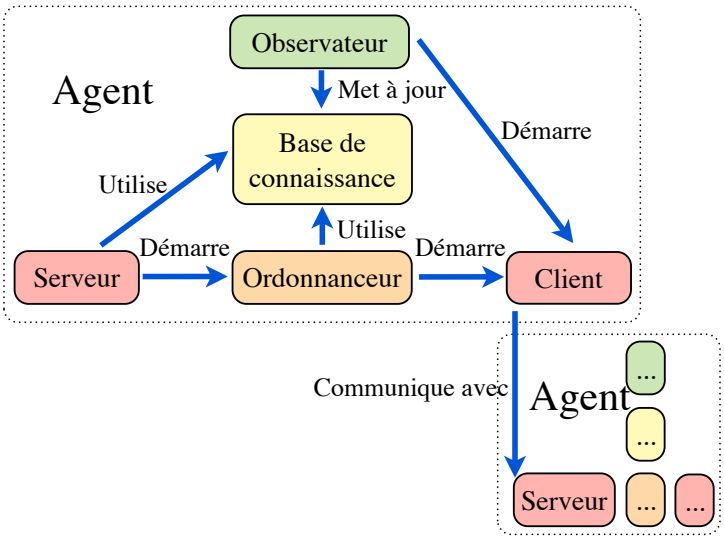


FIGURE 6.7 – Architecture d'un agent et interactions entre ses différents composants

Base de connaissance

La base de connaissance contient plusieurs informations.

Certaines informations sont disponibles de manière permanente, comme celles relatives :

- aux ressources du nœud ;
- aux machines virtuelles locales et à leur consommation en ressources ;
- aux seuils de surutilisation et de sous-utilisation ;
- à l'adresse IP et au port pour contacter l'agent du nœud voisin.

D'autres informations ne sont disponibles que pendant le traitement d'un événement et concernent notamment :

- l'initiateur ;
- le meneur actuel de la partition ;
- le raccourci (adresse IP et numéro de port) pour contacter l'agent du premier nœud en dehors de la partition ;
- l'état de la partition.

Le meneur de la partition dispose en plus des informations sur l'état des ressources des nœuds de cette partition.

Observateur

L'observateur met à jour de manière périodique (par défaut, toutes les deux secondes) le contenu de la base de connaissance concernant les machines virtuelles hébergées et leur consommation en ressources.

S'il détecte un problème et que le nœud n'est pas déjà impliqué dans une partition, il génère un événement, réserve le nœud pour résoudre cet événement et transfère ce dernier à l'agent du nœud voisin en ayant recours aux services d'un client.

Client

Un client est démarré à chaque fois qu'un message doit être envoyé à l'agent d'un autre nœud.

Un message est un objet Java qui est communiqué à un serveur à travers le réseau au moyen d'un connecteur logiciel (ou *socket*).

Serveur

Un serveur traite les messages en provenance des clients.

Dans le cas d'un initiateur, il peut être amené à vérifier que l'événement généré est toujours valide.

Dans le cas d'un meneur, il gère les requêtes de fusion provenant d'autres partitions.

Dans le cas d'un nœud quelconque, il :
- étudie les demandes de réservation et de dé-réservation du nœud ;
- met à jour la base de connaissance, lors d'un changement de meneur, avec les informations relatives (i) au nouveau meneur, (ii) à l'agent du premier nœud situé en dehors de la partition, et (iii) à l'état de la partition ;
- envoie au meneur, sur demande de ce dernier, les informations concernant les machines virtuelles hébergées et les ressources consommées ;
- met à jour la base de connaissance concernant la liste des machines virtuelles hébergées, en cas de migration ;
- décide quoi faire d'un événement qui lui est envoyé, soit le transférer à l'agent du premier nœud hors de sa partition, soit lancer un ordonnanceur pour tenter de résoudre l'événement.

Ordonnanceur

Un ordonnanceur commence par vérifier que l'événement qu'il cherche à résoudre est toujours valide ; il contacte pour cela l'initiateur.

Si l'événement n'est plus valide, l'ordonnanceur détruit la partition pour libérer les nœuds.

Si l'événement est au contraire toujours valide, l'ordonnanceur tente de calculer un plan de reconfiguration. Pour cela, il fait appel aux algorithmes d'ordonnancement d'un ordonnanceur externe ; dans le cadre de cette thèse, nous avons utilisé les algorithmes qui ont été conçus pour le gestionnaire Entropy, sur lequel nous reviendrons dans la section suivante ; toutefois, il serait possible de recourir à d'autres algorithmes.

Si aucun plan de reconfiguration n'est trouvé, l'ordonnanceur transfère l'événement à l'agent du premier nœud situé en dehors de la partition.

Si un plan est trouvé, il est appliqué ; une fois que l'événement est résolu, l'ordonnanceur détruit la partition.

6.2.2 Mise à profit des algorithmes d'ordonnancement du gestionnaire Entropy

Entropy [HLM+09, HLM10, HDL11] est un gestionnaire centralisé et autonome qui réalise de l'ordonnancement dynamique de machines virtuelles dans des centres de données et de calculs.

Entropy repose sur la programmation par contraintes pour calculer un plan de reconfiguration, ce qui présente plusieurs avantages. Cela permet notamment de trouver une solution optimale ou proche de l'optimal (suivant le temps alloué au calcul du plan de reconfiguration). Par ailleurs, il est possible de définir des algorithmes d'ordonnancement génériques et d'ajouter autant de contraintes que nécessaire (sur le placement des machines virtuelles par exemple), sans avoir besoin de modifier les algorithmes à chaque fois.

Entropy dispose de plusieurs caractéristiques particulièrement intéressantes dans le cadre de nos travaux autour de DVMS. Tout d'abord, Entropy met en œuvre plusieurs algorithmes d'ordonnancement de machines virtuelles ; la première version d'Entropy est surtout connue pour son algorithme de consolidation [HLM+09], tandis que la deuxième se concentre sur la correction des problèmes de violation de contraintes et apporte des gains importants en matière de passage à l'échelle [HDL11]. De plus, Entropy cherche à minimiser le nombre de migrations de machines virtuelles, étant donné que ce sont des opérations coûteuses. En outre, Entropy est capable de gérer des *vjobs*, autrement dit des groupes de machines virtuelles inter-dépendantes [HLM10]. Entropy permet également de définir des contraintes de placement liées aux affinités et aux antagonismes (i) entre machines virtuelles, ou (ii) entre des machines virtuelles d'une part et des nœuds d'autre part. Par ailleurs, le temps de calcul est très faible lorsqu'il n'y a pas de solution à un problème, ce qui permet d'accélérer le processus itératif de traitement des événements qui est mis en œuvre par DVMS. Enfin, Entropy est écrit en Java, ce qui facilite son intégration à DVMS.

6.3 Conclusion

Dans ce chapitre, nous avons détaillé une nouvelle proposition, DVMS (*Distributed Virtual Machine Scheduler*), pour faire de l'ordonnancement dynamique décentralisé de machines virtuelles dans les centres de données et de calculs.

Nous avons dans un premier temps présenté les principaux concepts qui constituent les fondements de DVMS [QL12], à savoir : la répartition du travail d'ordonnancement entre des agents installés sur les nœuds de calcul et organisés selon un anneau, le déclenchement d'un processus d'ordonnancement à l'apparition d'événements dans l'infrastructure, le partitionnement dynamique de l'infrastructure pour éviter les conflits entre ordonnanceurs, et l'adaptation des tailles des partitions à la complexité des événements à traiter, le tout sans recourir au moindre nœud de service.

Nous avons ensuite suggéré deux améliorations de l'algorithme responsable du traitement des événements [QLS12]. La première permet d'accélérer le parcours de l'anneau en faisant en sorte que chaque nœud d'une partition ait connaissance en permanence du premier nœud à l'extérieur de sa partition. La seconde amélioration consiste à garantir qu'un événement sera toujours résolu si une solution existe ; afin de gérer les pénuries de nœuds libres (résultant de la création d'un grand nombre

de partitions ou de l'agrandissement de certaines d'entre elles), nous avons proposé de fusionner les partitions deux à deux ; ceci permet d'offrir plus de possibilités de migrations de machines virtuelles à l'ordonnanceur travaillant sur la partition fusionnée.

Par la suite, nous avons expliqué comment nous avons mis en œuvre ces concepts dans un prototype programmé en Java. Nous avons indiqué que DVMS pouvait exploiter différents algorithmes d'ordonnancement pour calculer un plan de reconfiguration sur chaque partition. Dans le cadre de cette thèse, nous avons choisi de réutiliser ceux qui ont été conçus pour Entropy [HLM+09, HLM10, HDL11], un gestionnaire de machines virtuelles centralisé.

Nous verrons par la suite que le bon fonctionnement de l'approche a été validé au travers de différentes expériences. Nous étudierons enfin les limites de l'approche et les perspectives d'extension et d'amélioration.

7

Protocole expérimental et environnement de test

Après avoir détaillé les concepts et la mise en œuvre de DVMS, nous nous attachons dans ce chapitre à présenter la méthodologie et les outils qui nous ont permis de vérifier son bon fonctionnement.

Nous décrivons tout d'abord le protocole expérimental que nous avons suivi. Nous nous intéressons ensuite au logiciel que nous avons développé pour tester DVMS au travers de simulations et de mises en situation réelle sur le banc de test Grid'5000 [gri13]. Nous expliquons enfin l'utilisation que nous avons faite de la boîte à outils SimGrid [CLQ08] et en quoi elle nous a permis de nous affranchir des limitations rencontrées lors des expériences avec le logiciel de test.

Sommaire

7.1 Protocole expérimental

Pour pouvoir évaluer DVMS, nous avons défini un protocole expérimental, que nous détaillons ci-après.

7.1.1 Choix d'une plate-forme de test

Nous choisissons tout d'abord une plate-forme sur laquelle réaliser les expériences. Cette plate-forme peut être réelle (comme dans le cas du banc de test Grid'5000) ou simulée (lorsque la boîte à outils SimGrid est utilisée).

7.1.2 Définition des caractéristiques de l'expérience

Une fois la plate-forme de test choisie, nous définissons les caractéristiques de l'expérience à mener.

Nous choisissons tout d'abord le nombre de nœuds et les capacités en ressources de chacun. Seules les ressources processeur et mémoire sont prises en compte, mais il serait intéressant d'ajouter les dimensions réseau et disque.

Nous spécifions ensuite le nombre de machines virtuelles et leur consommation maximale en ressources physiques. Là encore, seules les ressources processeur et mémoire sont considérées.

Notons que le nombre de nœuds et de machines virtuelles reste constant au cours de chacune de nos expériences. Il serait cependant envisageable de faire varier le nombre de machines virtuelles au cours d'une même expérience (par exemple pour étudier la montée en charge de l'infrastructure). En revanche, à l'heure actuelle, le prototype DVMS ne permet pas de gérer l'ajout ou le retrait de nœuds de l'infrastructure.

7.1.3 Initialisation du système

Une fois les caractéristiques de l'expérience fixées, nous initialisons le système.

Dans le cas d'une plate-forme réelle, nous configurons tout d'abord les nœuds.

Dans le cas d'une mise en situation réelle, nous créons ensuite les machines virtuelles.

Enfin, dans tous les cas, nous configurons et nous démarrons l'ordonnanceur à évaluer (en l'occurrence DVMS ou Entropy [HLM+09, HLM10, HDL11], l'objectif étant de comparer les deux).

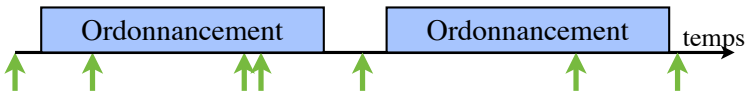
7.1.4 Injection de charge

Une fois que l'ordonnanceur à évaluer est démarré, nous injectons de la charge de travail processeur dans les machines virtuelles pour observer son comportement.

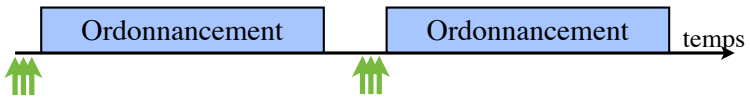
La charge à injecter peut être calculée grâce à une fonction aléatoire (une graine est utilisée afin d'assurer la reproductibilité des expériences) ou bien en lisant un fichier de trace.

L'injection de charge est réellement effectuée dans le cas d'une mise en situation réelle sur Grid'5000. En revanche, elle est simulée dans les autres cas.

L'injection de charge est étalée dans le temps pour les simulations avec SimGrid (cf. figure 7.1(a)) : la charge de travail est susceptible de varier pendant que l'ordonnanceur optimise le placement des machines virtuelles, ce qui reflète le fonctionnement d'un système réel. Dans les autres cas (c'est-à-dire avec le logiciel de test), la charge reste constante pendant le travail de l'ordonnanceur (cf. figure 7.1(b)) ; le but est alors d'observer la capacité de l'ordonnanceur à corriger un ensemble de problèmes donné.



(a) Injection de charge étalée dans le temps



(b) Injection de charge ponctuelle

FIGURE 7.1 – Différents types d'injection de charge utilisés

7.1.5 Traitement des résultats

Enfin, une fois l'expérience terminée, nous récupérons les résultats et nous les mettons en forme avec un outil d'analyse statistique (R) afin de les analyser.

7.2 Logiciel de test

Comme nous l'avons évoqué précédemment, nous avons développé un logiciel en Java afin de pouvoir tester le prototype DVMS dès les premières étapes de sa conception.

Ce logiciel permet principalement d'automatiser le déploiement des agents DVMS, d'injecter de la charge de travail et de récupérer les résultats des tests.

7.2.1 Configuration

Le logiciel de test est configurable. L'utilisateur peut notamment spécifier :
- le type d'expérience : simulation ou mise en situation réelle ;
- les nœuds utilisés pour l'expérience ;
- le nombre de machines virtuelles à démarrer et leur consommation maximale en ressources processeur et mémoire ;
- le nombre de valeurs différentes que peut prendre la consommation processeur d'une machine virtuelle ;
- le nombre de changements de la charge processeur d'une machine virtuelle au cours de l'expérience ;
- la façon d'obtenir la charge processeur à injecter dans chaque machine virtuelle :
  - via un fichier contenant la spécification de la charge processeur pour chaque machine virtuelle ;
  - via une fonction qui calcule une charge processeur aléatoire pour chaque machine virtuelle ; soient n le nombre de valeurs différentes que peut prendre la charge processeur, M la valeur maximale de la charge processeur, et i un entier choisi aléatoirement entre 0 et $n-1$, alors la charge processeur de la machine virtuelle sera égale à $M * i/(n-1)$; une graine est utilisée afin d'obtenir des résultats reproductibles.

Dans le cas d'une simulation, il faut en plus préciser le nombre de nœuds à simuler et leur capacité en ressources processeur et mémoire.

7.2.2 Composants

Le logiciel de test est constitué de trois composants principaux : une base de connaissance, un serveur et un pilote (cf. figure 7.2).

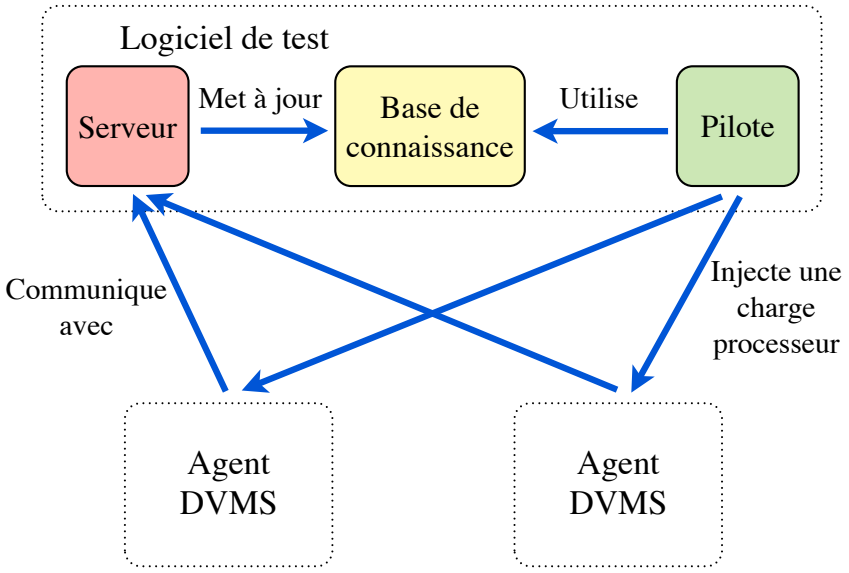


FIGURE 7.2 – Architecture du logiciel de test et interactions entre ses différents composants et les agents DVMS

Base de connaissance

La base de connaissance contient notamment :
- les différents paramètres spécifiés par l'utilisateur et qui ont été présentés précédemment ;
- les capacités processeur et mémoire de chaque nœud ;
- la consommation processeur et mémoire de chaque machine virtuelle ;
- la liste des machines virtuelles hébergées par chaque nœud ;
- l'information selon laquelle les agents sont autorisés à générer de nouveaux événements ou non ;
- la liste des événements en cours de traitement.

Serveur

Le serveur traite des messages en provenance des agents DVMS qui souhaitent avertir que :
- un nouvel agent DVMS est opérationnel ;
- une nouvelle partition est en cours de création ;
- une partition est détruite ;
- un événement lié à la surutilisation d'un nœud est résolu.

Pilote

Le pilote dirige l'expérience.

Au début d'une expérience, le pilote :

1. décide à quel nœud affecter chaque machine virtuelle ; les machines virtuelles sont assignées aux nœuds au moyen d'un tourniquet ;
2. démarre les machines virtuelles, dans le cas d'un déploiement sur système réel ;
3. décide sur quel nœud démarrer chaque agent DVMS ; les agents sont assignés aux nœuds au moyen d'un tourniquet ;
4. démarre les agents DVMS ;
5. attend que tous les agents DVMS soient opérationnels.

Ensuite, tant que l'expérience n'est pas terminée, le pilote répète la même série d'opérations ; dans l'ordre, le pilote :

1. calcule ou récupère dans un fichier (selon le paramétrage de l'utilisateur) la nouvelle valeur de la charge processeur de chaque machine virtuelle ;
2. communique à chaque nœud la charge processeur de chacune des machines virtuelles qu'il héberge ;
3. injecte la charge à chaque machine virtuelle, dans le cas d'un déploiement sur système réel ;
4. autorise les agents DVMS à générer des événements ;
5. attend que tous les événements liés à la surutilisation d'un nœud aient été résolus ;
6. interdit aux agents DVMS de générer de nouveaux événements ;
7. attend que toutes les partitions aient été détruites ;
8. écrit des informations dans un fichier tabulé concernant :
   - le nombre d'événements injectés ;
   - le temps qui s'est écoulé entre l'injection de charge et la résolution de tous les problèmes ;
   - le nombre de nœuds qui hébergent au moins une machine virtuelle.

Enfin, lorsque l'expérience se termine, le pilote :

1. récupère les fichiers écrits par les agents DVMS et contenant des informations sur chaque ordonnancement qui a eu lieu :
   - l'événement traité ;
   - le nombre de nœuds impliqués dans la partition ;
   - le résultat de l'ordonnancement (succès ou échec) ;
   - le temps de calcul d'un plan de reconfiguration ;
   - le temps d'application du plan de reconfiguration ;
2. concatène ces résultats dans un fichier texte tabulé.

Les fichiers tabulés contenant les résultats de l'expérience sont ensuite soumis à une analyse statistique au travers d'un script écrit dans le langage R. Les résultats de l'analyse sont à leur tour exportés dans un fichier texte tabulé.

Un logiciel de test similaire a été mis en œuvre pour piloter Entropy, avec pour objectif de comparer le comportement de ce dernier avec DVMS.

7.3 Grid'5000

Le banc de test Grid'5000 a été utilisé pour déployer DVMS avec le logiciel de test, afin d'effectuer des simulations et des mises en situation réelle.

7.3.1 Présentation

Grid'5000 est une grille « conçue pour soutenir la recherche expérimentale dans tous les domaines de l'informatique liés au calcul parallèle, large échelle ou réparti, et au réseau » [gri13].

La grille Grid'5000 est constituée à l'heure actuelle de dix sites (un site est équivalent à un centre de calculs), dont neuf sont situés en France et un au Luxembourg. Elle totalise vingt-cinq grappes et près de mille deux cents nœuds aux caractéristiques variées, en termes de processeur, de mémoire, de stockage et de réseau.

7.3.2 Simulations

Grid'5000 a dans un premier temps servi à tester DVMS au travers de simulations.

La procédure à suivre pour effectuer une simulation nécessite au préalable de :

1. réserver des nœuds (via `oarsub` [CDCG+05]) ;
2. déployer (via `kadeploy3` [kad13]) une distribution Linux Debian sur chacun de ces nœuds ;
3. installer les logiciels nécessaires à la simulation :
   - le logiciel de test et Entropy, sur un nœud de service ;
   - DVMS, sur chacun des nœuds de calcul ;
   - *Java Runtime Environment* (JRE), sur tous les nœuds, pour faire fonctionner les programmes mentionnés précédemment, qui sont écrits en Java ;
   - *Network Time Protocol* (NTP), pour synchroniser l'heure sur tous les nœuds ;
   - `sentinelle.pl`, un script permettant d'exécuter rapidement une même commande sur un grand nombre de nœuds, en l'occurrence pour permettre au logiciel de test de créer ou de supprimer des fichiers sur les nœuds de calcul ;
   - Taktuk [tak13], afin d'échanger rapidement des fichiers entre le nœud de service et tous les nœuds de calcul ;
4. configurer les agents DVMS et le logiciel de test ;
5. synchroniser (via `ntpdate`) l'horloge de tous les nœuds.

Une simulation peut ensuite être démarrée ; à son issue, les résultats sont récupérés.

La plus grande simulation réalisée sur Grid'5000 a impliqué 65 536 machines virtuelles simulées, ainsi que 10 240 agents DVMS répartis sur 28 nœuds de la grappe Suno, qui est rattachée au site de Sophia.

7.3.3 Mises en situation réelle

Grid'5000 a été utilisé dans un deuxième temps pour valider le bon fonctionnement de DVMS en situation réelle, une fois que le prototype a été plus mature.

La phase de préparation d'une mise en situation réelle est similaire à celle d'une simulation, sauf qu'il faut en plus :

1. réserver (via `oarsub`) un VLAN global et une plage d'adresses IP, pour pouvoir communiquer avec les machines virtuelles ;

2. installer un hyperviseur sur chaque nœud de calcul, pour pouvoir créer des machines virtuelles ; notre choix s'est porté sur KVM [KKL+07] ;
3. installer Libvirt [lib13] sur chaque nœud de calcul, pour faciliter l'utilisation de KVM ;
4. copier (via Taktuk) une image disque de machine virtuelle sur chaque nœud de calcul ; cette image disque est destinée à être partagée entre toutes les machines virtuelles hébergées sur un même nœud ; l'image disque est en lecture seule, les modifications apportées par une machine virtuelle sont stockées dans un fichier qui est propre à cette dernière ; lorsqu'une machine virtuelle est migrée, ce fichier est automatiquement transféré par KVM ;
5. créer (via Libvirt) chaque machine virtuelle sur le nœud approprié et configurer le réseau des machines virtuelles.

Ces opérations préliminaires sont effectuées par l'outil Flauncher [BCAC+12], qui a été conçu dans le cadre de l'initiative Hemera [hem13].

Au cours de l'expérience, le logiciel de test doit :

1. injecter une véritable charge de calcul dans chaque machine virtuelle (via `stress`) ;
2. migrer pour de bon les machines virtuelles (via Libvirt) lors de l'application d'un plan de reconfiguration.

La plus grande expérience réalisée a impliqué le déploiement de 4 754 machines virtuelles sur 467 nœuds répartis sur 4 sites (Nancy, Rennes, Sophia et Toulouse) et 8 grappes. Le routage de toutes les communications réseau était assuré par KaVLAN [kav13].

7.4 SimGrid

La boîte à outils SimGrid [CLQ08] a été utilisée dans un troisième temps pour compléter les expériences menées sur Grid'5000.

7.4.1 Présentation

SimGrid est « une boîte à outils qui fournit des fonctionnalités clés pour la simulation d'applications distribuées dans des environnements hétérogènes. Le but principal de ce projet est de faciliter la recherche dans le domaine des systèmes large échelle parallèles et distribués, tels que les grilles, les systèmes pair à pair et les centrales numériques » [sim13].

SimGrid propose quatre interfaces de programmation pour concevoir des applications distribuées :

1. SMPI permet de simuler le fonctionnement de programmes MPI non modifiés (seule une recompilation est nécessaire) ;
2. SimDag est utilisée pour modéliser le comportement d'un ensemble de tâches sous la forme d'un graphe acyclique orienté ;
3. GRAS (*Grid Reality And Simulation*) permet de développer des applications pour les grilles ; il existe deux mises en œuvre de cette interface ; la première permet de simuler le fonctionnement des applications, tandis que la seconde permet de les déployer sur un système réel ;

4. MSG est une interface simplifiée qui est utilisée pour la simulation d'applications constituées d'un ensemble de processus séquentiels concurrents.

7.4.2 Portage de DVMS sur SimGrid

MSG est l'interface de programmation la plus adaptée à l'architecture de DVMS ; afin de l'utiliser, nous avons dû effectuer plusieurs modifications sur DVMS et l'outil de test :

- transformer les fils d'exécution Java en *Process* SimGrid ;
- remplacer les communications réseau utilisant des connecteurs logiciels Java par des échanges de *Task* via des boîtes aux lettres SimGrid ;
- bannir l'usage des mécanismes de synchronisation Java sur la base de connaissance ; SimGrid s'assure en effet qu'il n'y ait pas d'accès concurrents, et le recours à la synchronisation peut nuire à son bon fonctionnement ;
- instrumenter le code pour tenir compte de l'écoulement du temps de simulation ;
- instrumenter le code pour récupérer des traces.

Nous en avons également profité pour réécrire la fonction d'injection de charge, afin d'étaler l'injection dans le temps.

7.4.3 Avantages par rapport aux simulations sur Grid'5000

En dépit des modifications nécessaires au portage de DVMS et du logiciel de test sur SimGrid, cette boîte à outils offre plusieurs avantages. SimGrid permet en effet de :

- simuler des infrastructures plus grandes ;
- s'affranchir des problématiques liées au déploiement d'une application sur une infrastructure distribuée (par exemple, la nécessité de synchroniser l'heure sur tous les nœuds, ou encore de gérer la défaillance de l'un des nœuds) ;
- répartir dans le temps l'injection de charge ;
- générer des traces qui peuvent être importées dans un de visualisation comme Pajé [paj13], ce qui est bien plus lisible que de parcourir les fichiers journaux de DVMS ; ceci a d'ailleurs permis de détecter et de corriger des bogues ;
- améliorer la reproductibilité des expériences ; avec SimGrid, l'ordonnancement des agents DVMS n'est par exemple plus tributaire de l'ordonnanceur de Linux.

7.4.4 Simulations

Une simulation avec SimGrid nécessite au préalable de :

1. paramétrer les agents DVMS et le logiciel de test ;
2. décrire l'infrastructure à simuler ; nous avons choisi de la modéliser sous la forme d'une grappe avec un routage complet entre tous les nœuds ;
3. décrire le déploiement ; le fichier de déploiement est généré automatiquement et sert à spécifier les paramètres de démarrage de chaque agent DVMS, qu'un agent doit être déployé sur chaque nœud de calcul et que le logiciel de test doit être déployé sur un nœud de service.

Une fois ceci fait, la simulation est démarrée. Lorsqu'elle est terminée, les traces sont récupérées et analysées avec Pajé.

La plus grande simulation réalisée concernait une exécution d'une heure impliquant 150 000 machines virtuelles réparties sur 10 000 nœuds. La consommation processeur de chaque machine virtuelle était modifiée toutes les 2 minutes.

7.5 Conclusion

Au cours de ce chapitre, nous avons présenté la méthodologie et les outils utilisés pour valider le bon fonctionnement de DVMS.

Nous nous sommes tout d'abord penchés sur le protocole expérimental.

Nous nous sommes ensuite intéressés au paramétrage et au fonctionnement d'un logiciel conçu spécialement pour tester DVMS.

Nous avons également décrit le déploiement de ce logiciel sur Grid'5000 [gri13], afin d'évaluer DVMS au travers de simulations, puis de mises en situation réelle (en nous appuyant dans ce dernier cas sur l'outil Flauncher [BCAC+12] pour configurer les nœuds et les machines virtuelles).

Nous avons enfin expliqué ce que la boîte à outils SimGrid [CLQ08] nous avait apporté en matière de simulations.

Dans le chapitre suivant, nous nous pencherons sur l'analyse des résultats obtenus grâce à ces différents outils.

8

Résultats expérimentaux et validation de DVMS

Afin de valider l'approche DVMS, nous l'avons comparée à celle mise en œuvre dans Entropy [HLM+09, HDL11].

La première version d'Entropy a pour objectif de consolider les machines virtuelles sur un nombre restreint de nœuds, tandis que la deuxième version permet de réparer l'infrastructure en corrigeant les problèmes de surutilisation de nœuds et les violations de contraintes de placement. Bien que ce changement de stratégie permette de diminuer de manière significative le temps de calcul d'un plan de reconfiguration [HDL11], Entropy prend toujours en compte l'ensemble des problèmes, ce qui a un impact sur le passage à l'échelle.

Afin d'évaluer en quoi DVMS permet d'améliorer le passage à l'échelle et la réactivité des politiques d'ordonnancement conçues pour Entropy, nous avons mené plusieurs expériences avec ces deux logiciels en nous appuyant sur le protocole expérimental et les environnements de test définis au chapitre précédent. Nous avons dans un premier temps effectué des simulations sur le banc de test Grid'5000 [gri13] ; nous avons par la suite testé DVMS et Entropy en situation réelle, toujours sur Grid'5000 ; enfin, nous avons réalisé des simulations avec l'outil SimGrid [CLQ08].

Dans ce chapitre, nous présentons les résultats des différentes expériences menées avec DVMS et Entropy.

Sommaire

8.1 Simulations sur Grid'5000

Nous avons mené deux séries de simulations sur Grid'5000 pour comparer Entropy et DVMS.

La première série a été réalisée avec l'algorithme de consolidation conçu pour Entropy 1, tandis que la seconde a eu pour objet l'algorithme de réparation d'infrastructure conçu pour Entropy 2.

Nous avons fait en sorte que, à chaque étape de chaque simulation, Entropy et DVMS travaillent à partir de la même configuration initiale : la répartition initiale des machines virtuelles sur les nœuds était la même, la charge de travail injectée aux machines virtuelles était identique, ce qui fait que les problèmes à résoudre étaient les mêmes.

Nous avons utilisé plusieurs critères pour comparer les deux approches. Un premier critère concernait la durée d'une itération, c'est-à-dire le temps écoulé entre le moment où le logiciel de test injectait la charge et celui où tous les problèmes de surutilisation de nœuds étaient corrigés ; notons que cette durée n'était influencée que par le temps de calcul des plans de reconfigurations ; en effet, nous avons considéré dans ces simulations que la collecte des informations sur l'état des nœuds et la migration des machines virtuelles étaient réalisées de manière instantanée. Un autre critère correspondait à la taille de la partition, autrement dit le nombre de nœuds à prendre en compte pour corriger un problème. Un troisième critère de comparaison important était le coût d'un plan de reconfiguration, qui donnait une estimation du temps d'application d'un tel plan ; cette information n'était cependant disponible que pour la première série de simulations ; en effet, Entropy 2 utilisait un modèle de coût qui ne permettait pas de comparer les deux approches ; étant donné que DVMS cherchait à limiter la taille des partitions, nous avons émis l'hypothèse que son comportement lors de la seconde série de simulations serait similaire à celui observé lors de la première.

Pour chaque série de simulations, nous détaillons les paramètres expérimentaux avant de présenter et d'analyser les résultats.

8.1.1 Consolidation

Paramètres expérimentaux

Nous avons réalisé la première série de simulations avec l'algorithme de consolidation conçu pour Entropy 1 sur un nœud HP Proliant DL165 G7 disposant de deux processeurs (AMD Opteron 6164 HE, 12 cœurs, 1.7 GHz) et de 48 Go de mémoire vive.

Notre environnement logiciel était constitué d'une distribution Linux Debian 6 (Squeeze) 64 bits, de OpenJDK JRE 6 et d'Entropy 1.1.1.

Les simulations ont porté sur des infrastructures comportant respectivement (i) 128 machines virtuelles réparties sur 64 nœuds et (ii) 256 machines virtuelles réparties sur 128 nœuds, cette dernière infrastructure correspondant à la limite d'Entropy 1 en matière de passage à l'échelle. Cette limitation a un impact direct sur DVMS, qui recourt à l'algorithme d'Entropy pour ordonnancer les machines virtuelles d'une partition.

Chaque simulation a été réalisée sur 30 itérations.

Les nœuds simulés comportaient deux processeurs à 2 GHz et 4 Go de mémoire vive. Les machines virtuelles simulées étaient dotées d'un processeur virtuel à 2 GHz et de 1 Go de mémoire vive. La consommation processeur d'une machine virtuelle pouvait prendre une des valeurs suivantes (en MHz) : 0, 400, 800, 1 200, 1 600 ou 2 000.

DVMS était configuré pour considérer qu'un nœud était :

- surutilisé si ses machines virtuelles cherchaient à consommer plus de 100 % de ses ressources processeur ou mémoire ;
- sous-utilisé si moins de 20 % de son processeur et moins de 50 % de sa mémoire étaient utilisés.

DVMS et Entropy étaient paramétrés pour que le temps de calcul d'un plan de reconfiguration ne dépasse pas quatre fois le nombre de nœuds pris en compte (en secondes). Notons que cette valeur a été déterminée de manière empirique.

Résultats

Les figures 8.1, 8.2, 8.3, 8.4, 8.5 et 8.6 précisent les tendances pour les principaux critères d'étude, en donnant à chaque fois une représentation graphique de la moyenne et de l'écart type des valeurs obtenues.

Le nombre moyen d'événements injectés à chaque itération est visible sur la figure 8.1.

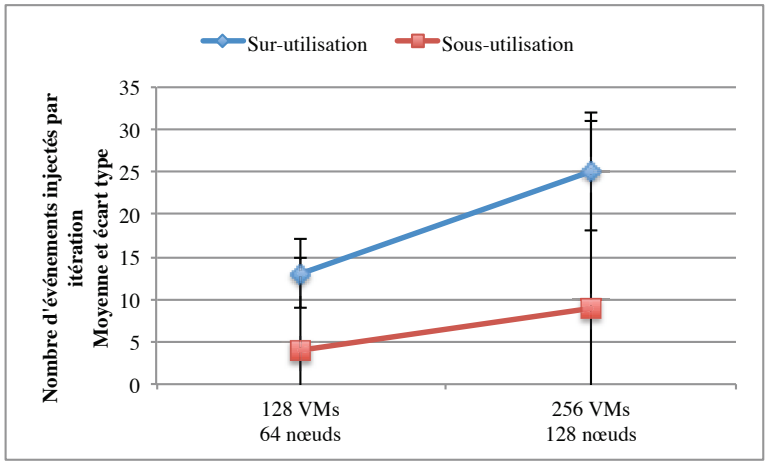


FIGURE 8.1 – Nombre moyen d'événements injectés à chaque itération avec la politique de consolidation

Nous pouvons tout d'abord observer sur la figure 8.2 que DVMS a permis de résoudre l'ensemble des événements plus rapidement qu'Entropy, jusqu'à près de cinq fois en moyenne.

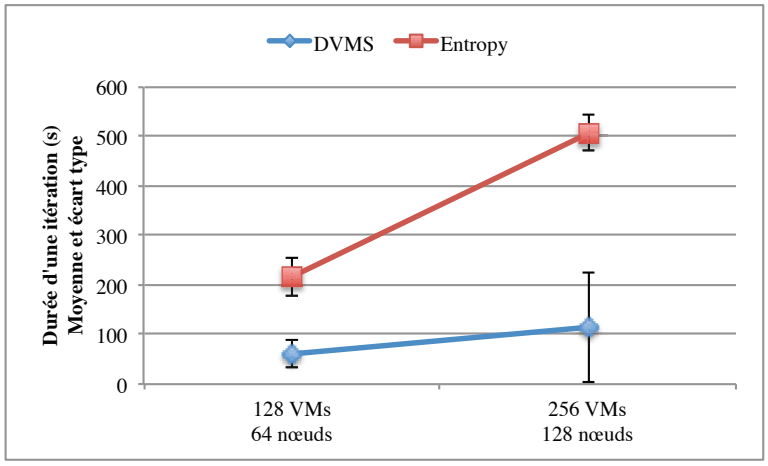


FIGURE 8.2 – Durée moyenne d'une itération avec la politique de consolidation

Cette rapidité de traitement ne s'est pas faite au détriment du taux de consolidation puisque DVMS n'a eu besoin en moyenne que de deux à trois nœuds de plus qu'Entropy pour héberger l'ensemble des machines virtuelles (cf. figure 8.3). Ce résultat n'a cependant pu être atteint qu'en spécifiant un seuil de sous-utilisation relativement élevé pour chaque nœud ; un nœud identifié comme étant sous-utilisé pouvait ne pas l'être en réalité, et cela prenait du temps pour trouver suffisamment de nœuds afin de le débarrasser de toutes ses machines virtuelles.

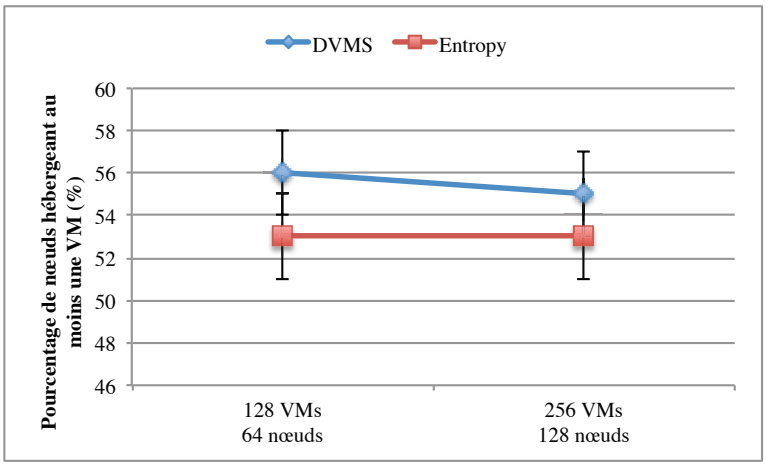


FIGURE 8.3 – Pourcentage moyen de nœuds hébergeant au moins une machine virtuelle avec la politique de consolidation

Considérons maintenant les résultats non plus à l'échelle de l'infrastructure, mais à l'échelle de la partition ; nous pouvons observer sur la figure 8.4 que DVMS a constitué des partitions de petite taille pour résoudre les événements : 5 nœuds en moyenne, et même seulement 3 nœuds en moyenne en tenant en compte uniquement des événements liés à la surutilisation d'un nœud. À l'inverse, Entropy a pris en compte tous les nœuds de l'infrastructure pour chaque calcul de plan de reconfiguration.

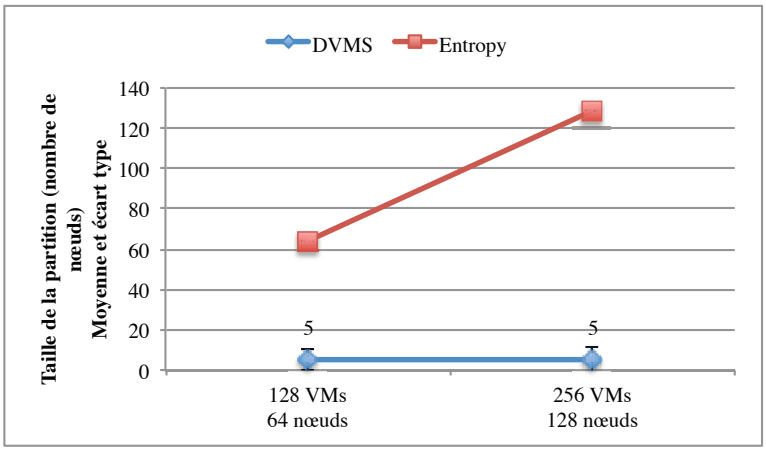


FIGURE 8.4 – Taille moyenne d'une partition avec la politique de consolidation

Plus petite était la partition, plus faible s'est révélé le temps nécessaire à la résolution d'un événement, c'est à dire le temps écoulé entre la détection de cet événement et sa résolution (cf. figure 8.5). Notons qu'Entropy résolvait tous les événements à la fois, c'est pourquoi le temps nécessaire à la résolution d'un événement était égal à la durée de l'itération correspondante (ceci n'était vrai que parce que nous nous étions placés dans le cas où l'application du plan de reconfiguration était instantanée).

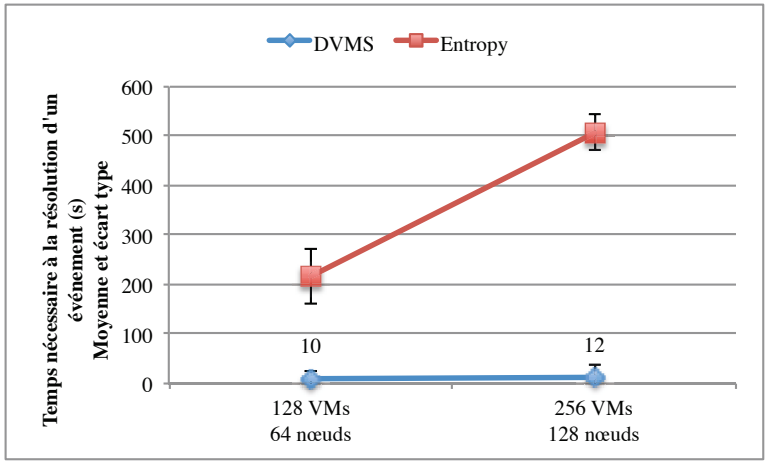


FIGURE 8.5 – Temps moyen nécessaire à la résolution d'un événement avec la politique de consolidation

En plus d'avoir influencé le temps de résolution d'un événement, la taille de la partition a eu une incidence sur le coût d'application du plan de reconfiguration : plus petite était la partition et plus faible était le coût d'application d'un plan (cf. figure 8.6) ; l'explication résidait dans le fait que plus la partition était petite et moins il y avait de machines virtuelles à migrer. Il est toutefois nécessaire de nuancer cette observation : un plan de reconfiguration d'Entropy permettait de résoudre tous les événements d'un coup, tandis que DVMS traitait les événements au cas par cas ; ainsi, le coût de l'ensemble des plans de reconfiguration de DVMS pouvait être supérieur à celui d'un plan de reconfiguration d'Entropy ; cependant, nous nous sommes concentrés avec DVMS sur la résolution de chaque événement le plus rapidement possible, c'est pourquoi nous n'avons pas considéré le coût global mais seulement le coût lié à chaque événement.

8.1.2 Réparation de l'infrastructure

Afin d'améliorer le passage à l'échelle, la nouvelle version d'Entropy [HDL11] s'applique uniquement à maintenir la viabilité de l'infrastructure en prenant périodiquement en compte les besoins en ressources des machines virtuelles et éventuellement différentes contraintes de placement spécifiées par l'administrateur.

L'algorithme de réparation conçu pour Entropy 2 n'essaie donc plus de consolider les machines virtuelles sur un nombre restreint de nœuds ; autrement dit, il n'effectue aucune action particulière en cas de sous-utilisation d'un nœud. C'est pourquoi nous avons désactivé dans DVMS la génération d'événements liés à la sous-utilisation d'un nœud.

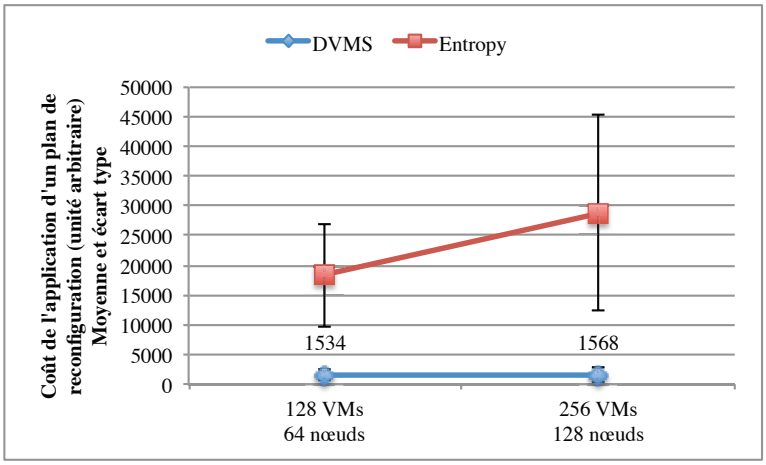


FIGURE 8.6 – Coût moyen d'application d'un plan de reconfiguration avec la politique de consolidation

Paramètres expérimentaux

Nous avons réalisé la seconde série de simulations avec l'algorithme de consolidation conçu pour Entropy 2 sur 3, 6 et 12 (respectivement pour la simulation de 2 048 machines virtuelles sur 320 nœuds, 4 096 machines virtuelles sur 640 nœuds et 8 192 machines virtuelles sur 1 280 nœuds) nœuds HP Proliant DL165 G7 disposant de deux processeurs (AMD Opteron 6164 HE, 12 cœurs, 1.7 GHz) et de 48 Go de mémoire vive.

Notre environnement logiciel était constitué d'une distribution Linux Debian 6 (Squeeze) 64 bits, de OpenJDK JRE 6 et d'Entropy 2.1.

Chaque simulation a été réalisée sur 30 itérations.

Les nœuds simulés comportaient quatre processeurs d'une capacité de 5 unités arbitraires chacun et 8 Go de mémoire vive. Les machines virtuelles simulées étaient dotées d'un processeur virtuel et de 1 Go de mémoire vive. La consommation processeur d'une machine virtuelle pouvait prendre une des valeurs suivantes (en unité arbitraire) : 0, 1, 2, 3, 4 ou 5.

DVMS était configuré pour considérer qu'un nœud était surutilisé si ses machines virtuelles cherchaient à consommer plus de 100 % de ses ressources processeur ou mémoire. Les événements liés à la sous-utilisation d'un nœud étaient désactivés, comme nous l'avons précédemment mentionné.

DVMS et Entropy étaient paramétrés pour que le temps de calcul d'un plan de reconfiguration ne dépasse pas un dixième du nombre de nœuds pris en compte (en secondes). Là encore, cette valeur a été déterminée de manière empirique.

Résultats

Les figures 8.7, 8.8, 8.9 et 8.10 précisent les tendances pour les principaux critères d'étude, en donnant à chaque fois une représentation graphique de la moyenne et de l'écart type des valeurs obtenues. Notons que les barres d'erreur ne sont pas toujours visibles, étant donné que les écarts types sont relativement faibles.

Le nombre moyen d'événements injectés à chaque itération est visible sur la figure 8.7.

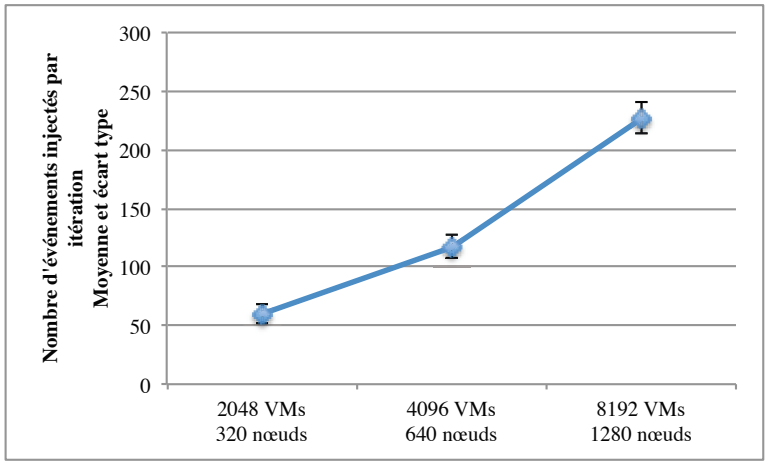


FIGURE 8.7 – Nombre moyen d'événements injectés à chaque itération avec la politique de réparation

La figure 8.8 présente la durée moyenne d'une itération. Nous remarquons tout d'abord que celle-ci s'est révélée bien inférieure à la durée moyenne d'une itération avec la politique de consolidation, bien que les infrastructures utilisées pour tester la politique de réparation aient été d'une taille bien supérieure (jusqu'à 10 fois plus de nœuds et 32 fois plus de machines virtuelles). Nous constatons également que DVMS a corrigé l'ensemble des problèmes 10 à 20 fois plus rapidement qu'Entropy.

Notons que chaque nœud de l'infrastructure hébergeait toujours plusieurs machines virtuelles, étant donné que la politique de réparation n'avait pas pour objectif de consolider les machines virtuelles et de libérer des nœuds. Le pourcentage moyen de nœuds hébergeant au moins une machine virtuelle n'est donc pas une information pertinente.

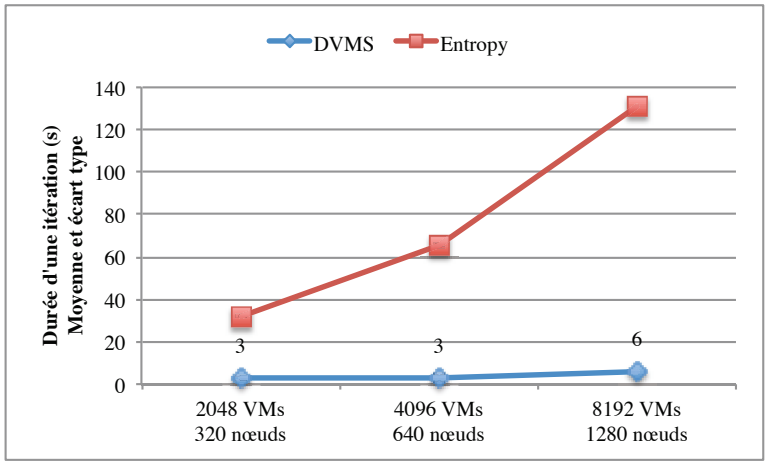


FIGURE 8.8 – Durée moyenne d'une itération avec la politique de réparation

La taille moyenne d'une partition nécessaire à la résolution d'un événement est demeurée très faible (cf. figure 8.9) : 3 nœuds, soit la même chose qu'avec la politique de consolidation.

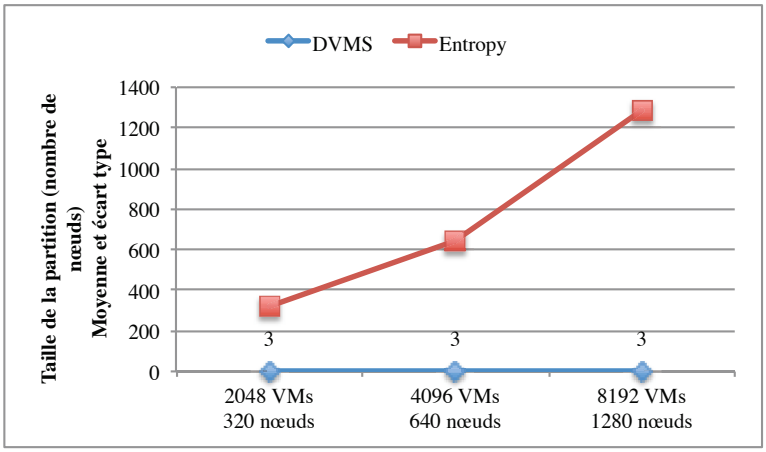


FIGURE 8.9 – Taille moyenne d'une partition avec la politique de réparation

Nous pouvons observer que le temps de résolution d'un événement (cf. figure 8.10) est une fois de plus corrélé à la taille de la partition associée.

Par ailleurs, nous constatons que DVMS s'est comporté particulièrement bien avec la politique de réparation et sans les événements liés à la sous-utilisation d'un nœud : le temps de résolution d'un événement s'est en effet avéré inférieur à une seconde en moyenne.

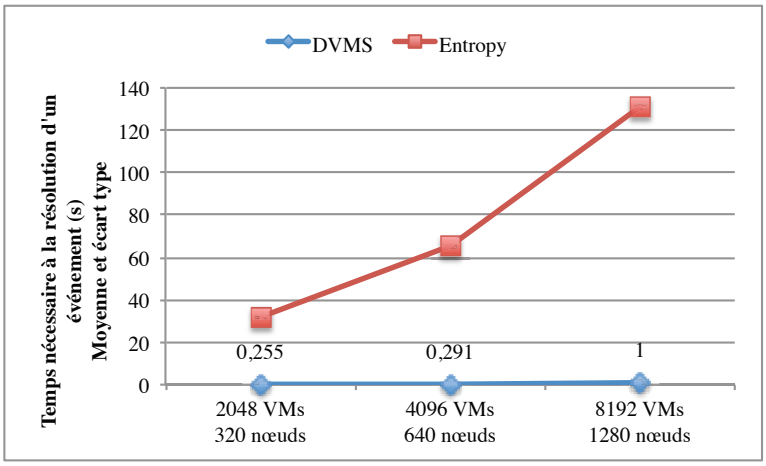


FIGURE 8.10 – Temps moyen nécessaire à la résolution d'un événement avec la politique de réparation

8.2 Mises en situation réelle sur Grid'5000

Les simulations sur Grid'5000 nous ont permis d'obtenir des informations intéressantes sur le comportement de DVMS et ses points forts par rapport à Entropy, en particulier quand il s'agissait de trouver rapidement une solution à un problème donné. En revanche, nous n'avons pas pu étudier en détail le coût d'application des plans de reconfiguration.

Afin de compléter notre évaluation, nous avons testé Entropy et DVMS en situation réelle sur Grid'5000.

8.2.1 Paramètres expérimentaux

Nous avons effectué les mises en situation réelle avec l'algorithme de réparation conçu pour Entropy 2.

Le nombre et les caractéristiques des nœuds de calcul utilisés sont visibles respectivement sur la figure 8.11 et le tableau 8.1. Le nœud de service était un nœud de la grappe Suno.

Notre environnement logiciel était constitué d'une distribution Linux Debian 6 (Squeeze) 64 bits, de OpenJDK JRE 6, de virt-install 0.600.1, de virsh 0.9.12, de KVM 1.1.2, de Flauncher et d'Entropy 2.1.

Chaque expérience a été réalisée sur 10 itérations.

Les machines virtuelles étaient dotées d'un processeur virtuel et de 1 Go de mémoire vive. La consommation processeur d'une machine virtuelle était égale à 0 ou à 100 % de son processeur virtuel.

DVMS était configuré pour considérer qu'un nœud était surutilisé si ses machines virtuelles cherchaient à consommer plus de 100 % de ses ressources processeur ou mémoire. Comme précédemment, les événements liés à la sous-utilisation d'un nœud étaient désactivés.

DVMS et Entropy étaient paramétrés pour que le temps de calcul d'un plan de reconfiguration ne dépasse pas un dixième du nombre de nœuds pris en compte (en secondes). Rappelons que cette valeur a été déterminée de manière empirique.

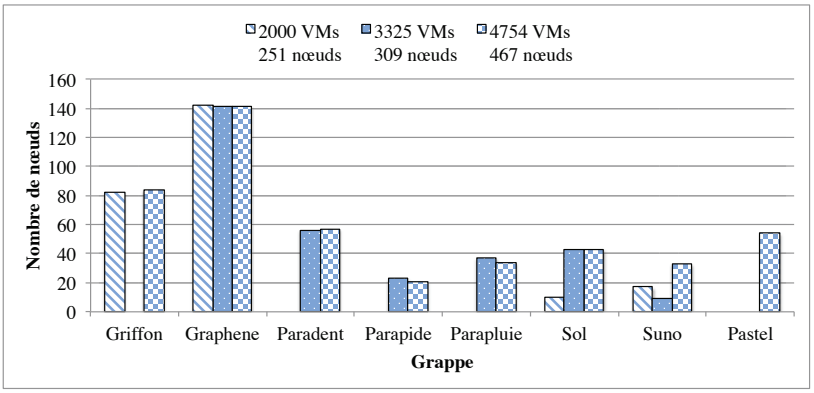


FIGURE 8.11 – Nœuds de calcul utilisés au cours des mises en situation réelle

Site	Grappe	Processeur	Mémoire vive (Go)
Nancy	Griffon	Intel ; 2x4 cœurs ; 2,50 GHz	16
	Graphene	Intel ; 1x4 cœurs ; 2,60 GHz	16
Rennes	Paradent	Intel ; 2x4 cœurs ; 2,50 GHz	32
	Parapide	Intel ; 2x4 cœurs ; 2,93 GHz	24
	Parapluie	AMD ; 2x12 cœurs ; 1,70 GHz	48
Sophia	Sol	AMD ; 2x2 cœurs ; 2,60 GHz	4
	Suno	Intel ; 2x4 cœurs ; 2,26 GHz	32
Toulouse	Pastel	AMD ; 2x2 cœurs ; 2,61 GHz	8

TABLEAU 8.1 – Caractéristiques des nœuds utilisés au cours des mises en situation réelle

8.2.2 Résultats

Les figures 8.12, 8.13, 8.14 et 8.15 précisent les tendances pour les principaux critères d'étude, en donnant à chaque fois une représentation graphique de la moyenne et de l'écart type des valeurs obtenues.

Le nombre moyen d'événements injectés à chaque itération est visible sur la figure 8.12.

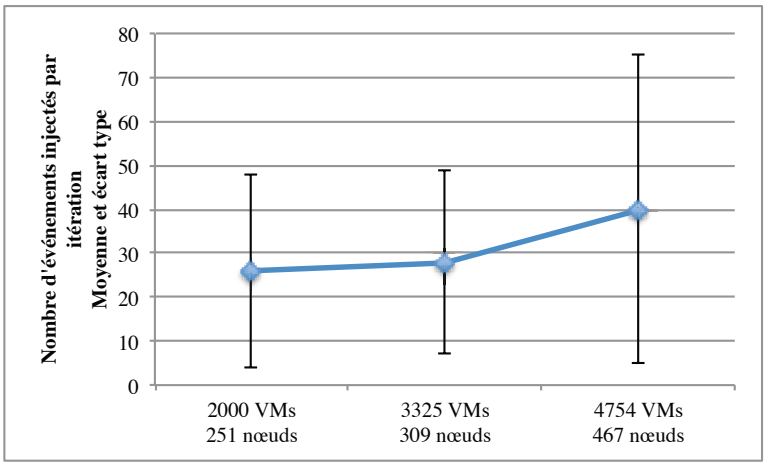


FIGURE 8.12 – Nombre moyen d'événements injectés à chaque itération avec la politique de réparation

Nous pouvons tout d'abord constater que la durée moyenne d'une itération (cf. figure 8.13) s'est notablement allongée par rapport aux simulations. Ce constat est valable aussi bien pour DVMS que pour Entropy. D'ailleurs, si l'écart entre les deux solutions est resté semblable en valeur absolue, il s'est réduit en valeur relative.

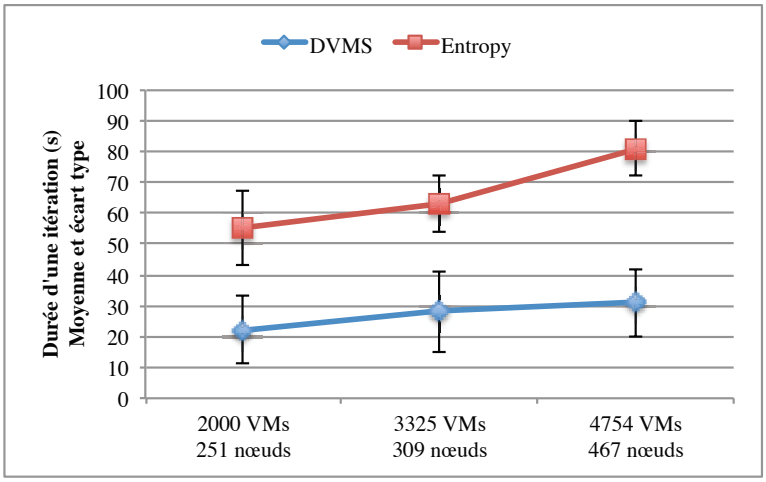


FIGURE 8.13 – Durée moyenne d'une itération avec la politique de réparation

Dans le cas de DVMS, l'allongement de la durée d'une itération s'explique par l'accroissement du temps nécessaire à la résolution de chaque événement (cf. figure 8.14). En effet, là où DVMS corrigeait les problèmes de surutilisation des ressources en moins d'une seconde en moyenne lors des simulations, il lui a fallu plus d'une dizaine de secondes pour les mises en situation réelle.

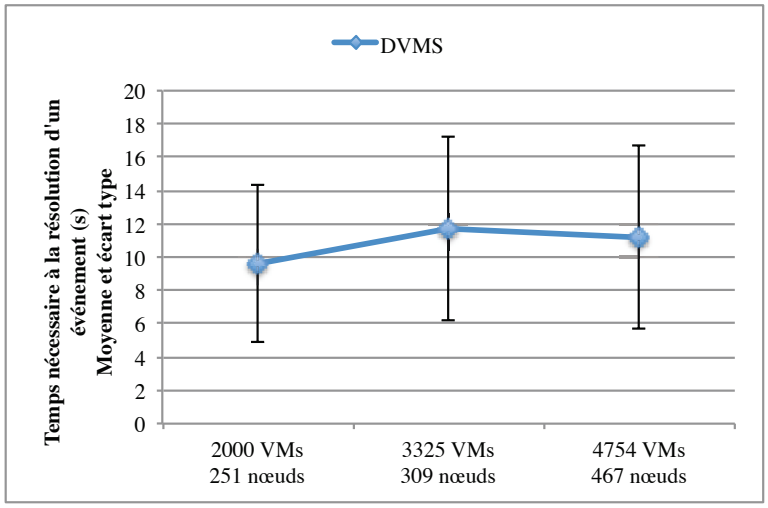


FIGURE 8.14 – Temps moyen nécessaire à la résolution d'un événement avec la politique de réparation

Si DVMS a mis plus de temps à résoudre chaque événement, c'est parce que l'application des plans de reconfiguration a été longue (cf. figure 8.15). En fait, dans le cas de DVMS, le temps de résolution d'un événement est largement dominé par le temps d'application du plan de reconfiguration (le calcul du plan est très rapide, ainsi que nous l'avons vu au travers des simulations). Entropy en revanche a passé à peu près autant de temps à calculer un plan de reconfiguration qu'à l'appliquer.

Notons au passage qu'Entropy et DVMS ont effectué exactement le même nombre de migrations, étant donné qu'ils devaient résoudre les mêmes problèmes de surutilisation des ressources.

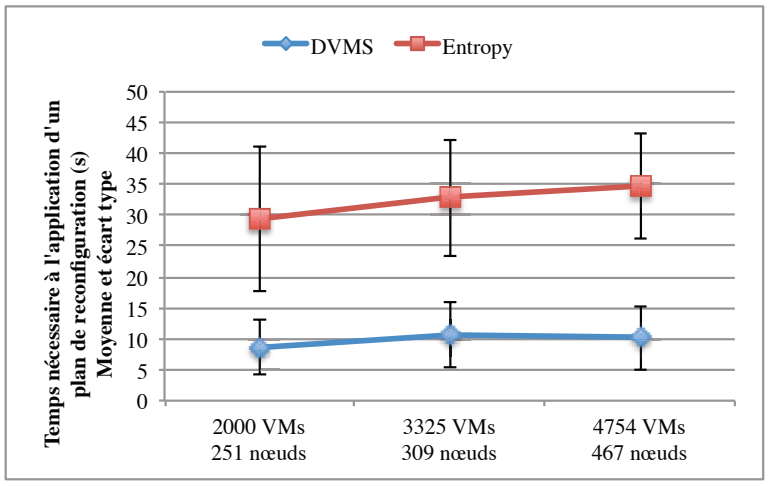


FIGURE 8.15 – Temps moyen nécessaire à l'application d'un plan de reconfiguration avec la politique de réparation

8.3 Simulations avec SimGrid

Les simulations et les mises en situation réelle sur Grid'5000 nous ont permis de constater respectivement que DVMS était capable de trouver très rapidement une solution à un problème de surutilisation des ressources, mais que l'application de cette solution prenait un certain temps.

En revanche, la façon dont nous avons injecté la charge de travail n'était pas représentative d'un système réel ; en effet, nous nous interdisions jusqu'à présent de modifier la charge tant que l'ordonnanceur n'avait pas terminé son travail.

Afin d'approfondir notre évaluation de DVMS et d'Entropy, nous avons donc mis en œuvre une injection de charge étalée dans le temps grâce à l'outil SimGrid.

8.3.1 Paramètres expérimentaux

Nous avons réalisé chaque simulation SimGrid avec l'algorithme de consolidation conçu pour Entropy 2 sur un nœud HP Proliant DL165 G7 disposant de deux processeurs (AMD Opteron 6164 HE, 12 cœurs, 1.7 GHz) et de 48 Go de mémoire vive.

Notre environnement logiciel était constitué d'une distribution Linux Debian 6 (Squeeze) 64 bits, de OpenJDK JRE 6, de SimGrid 3.7.1 et d'Entropy 2.1.

Les simulations ont porté sur des infrastructures comportant respectivement :

1. 5 120 machines virtuelles réparties sur 512 nœuds ;
2. 10 240 machines virtuelles réparties sur 1 024 nœuds ;
3. 20 480 machines virtuelles réparties sur 2 048 nœuds ;
4. 40 960 machines virtuelles réparties sur 4 096 nœuds ;
5. 81 920 machines virtuelles réparties sur 8 192 nœuds.

La durée simulée de chaque expérience était fixée à 1 heure.

Les nœuds simulés comportaient 8 processeurs d'une capacité de 100 unités arbitraires chacun et 20 Go de mémoire vive. Les machines virtuelles simulées étaient dotées d'un processeur virtuel et de 1 Go de mémoire vive. La consommation processeur d'une machine virtuelle pouvait prendre une des valeurs suivantes (en unité arbitraire) : 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 ou 100. La consommation processeur d'une machine virtuelle suivait une loi exponentielle dont l'espérance et l'écart type valaient 50 unités arbitraires ; en régime permanent, les machines virtuelles consommaient donc en moyenne 50 unités arbitraire de processeur ; les ressources processeur de l'infrastructure simulée étaient par conséquent utilisées à 62,5 %. La consommation processeur d'une machine virtuelle variait en moyenne toutes les 5 minutes.

DVMS était configuré pour considérer qu'un nœud était surutilisé si ses machines virtuelles cherchaient à consommer plus de 100 % de ses ressources processeur ou mémoire. Comme précédemment, les événements liés à la sous-utilisation d'un nœud étaient désactivés.

DVMS vérifiait l'état des ressources de chaque nœud toutes les secondes. Entropy a dans un premier temps été configuré pour attendre une seconde après la fin d'un calcul d'ordonnancement avant d'en commencer un nouveau ; le délai a ensuite été porté à 30 secondes.

DVMS et Entropy étaient paramétrés pour que le temps de calcul d'un plan de reconfiguration ne dépasse pas un quart du nombre de nœuds pris en compte (en

secondes). Notons que, comme pour les simulations sur Grid'5000, les plans de reconfiguration étaient appliqués instantanément.

8.3.2 Résultats

La figure 8.16 présente le temps de calcul cumulé pour DVMS et Entropy, en différenciant le cas où Entropy était relancé une seconde après avoir terminé un ordonnancement du cas où il était relancé après 30 secondes. Nous pouvons voir sur cette figure qu'Entropy a passé beaucoup de temps à calculer des plans de reconfiguration ; en fait, dès que la taille de l'infrastructure à gérer augmentait, il travaillait pendant pratiquement toute la durée de la simulation. En apparence, DVMS a également passé un certain temps à calculer ; cependant, en ramenant ce temps de calcul au nombre de nœuds, nous nous apercevons qu'il était très faible, de l'ordre de quelques dixièmes de seconde par nœud en moyenne pour une heure de simulation.

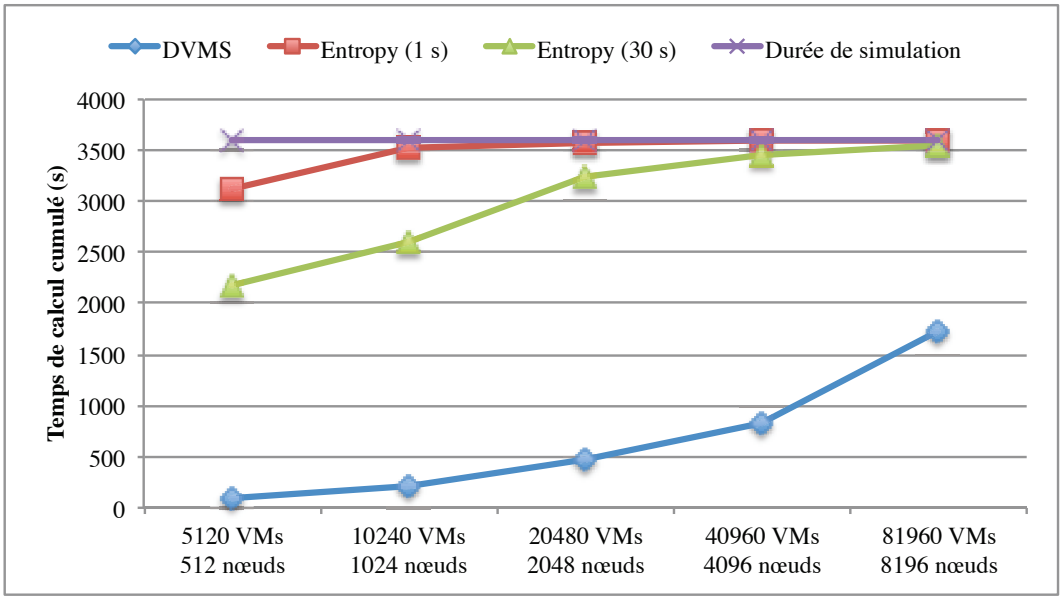


FIGURE 8.16 – Temps de calcul cumulé

Nous savions déjà que DVMS réagissait rapidement pour corriger les problèmes de surutilisation des ressources ; nous en avons la confirmation au travers de la figure 8.17. En effet, le temps de surutilisation cumulé sur une heure de simulation s'est élevé à moins de 12 minutes pour DVMS, ce qui correspond en moyenne à moins d'un dixième de seconde de surutilisation des ressources de chaque nœud. Ces résultats sont cependant à nuancer, étant donné que les plans de reconfiguration sont appliqués de manière instantanée.

La figure 8.17 nous présente en outre un résultat plus surprenant : l'action d'Entropy n'a pas été perceptible avec les paramètres expérimentaux que nous avons retenus. Il semblerait que la consommation en ressources des machines virtuelles ait fluctué durant le calcul des plans de reconfiguration, rendant ces derniers caducs. Autrement dit, le temps nécessaire à Entropy pour calculer un plan de reconfiguration était supérieur à la durée moyenne de surutilisation des noeuds.

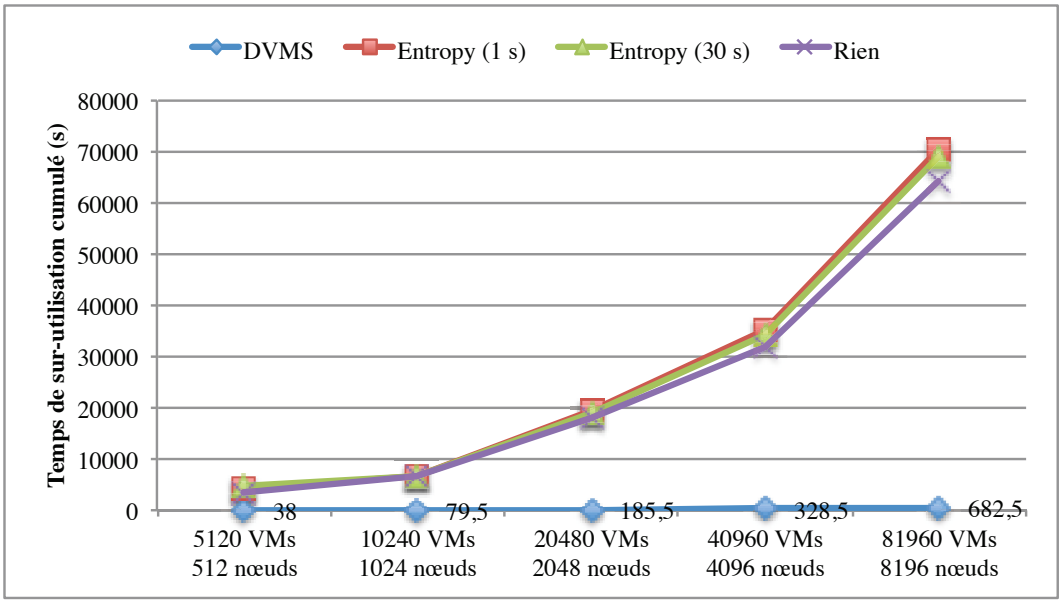


FIGURE 8.17 – Temps de surutilisation cumulé

8.4 Conclusion

Au cours de ce chapitre, nous avons étudié les résultats expérimentaux obtenus au cours des tests réalisés avec Entropy [HLM+09, HDL11] et DVMS.

Nous avons constaté au travers de l'analyse des simulations effectuées sur Grid'5000 [gri13] que DVMS constituait généralement des partitions de petite taille, ce qui lui permettait de trouver très rapidement une solution à un problème de surutilisation (ou de sous-utilisation) des ressources d'un nœud. Par ailleurs, cette rapidité de traitement ne s'est pas faite au détriment du taux de consolidation, puisque DVMS a soutenu la comparaison avec Entropy sur ce point.

Nous avons ensuite complété ces évaluations par des mises en situation réelle sur Grid'5000, afin d'observer le coût d'application des plans de reconfiguration. Nous avons ainsi constaté que, dans le cas de DVMS, le temps d'application d'un plan dominait largement le temps nécessaire au calcul de ce plan ; Entropy a passé quant à lui autant de temps à réaliser chacune de ces opérations.

Enfin, nous avons utilisé l'outil SimGrid [CLQ08] afin d'observer le comportement d'Entropy et de DVMS lorsque la charge de travail des machines virtuelles fluctuait pendant qu'ils étaient en train de calculer un plan de reconfiguration. Nous avons ainsi observé que DVMS consommait peu de ressources processeur sur chacun des nœuds. Nous avons également eu la confirmation qu'il était apte à corriger rapidement les problèmes de surutilisation des ressources d'un nœud. En revanche, nous avons constaté que ces paramètres expérimentaux pénalisaient Entropy, dans la mesure où les plans de reconfiguration devenaient caducs avant de pouvoir être appliqués.

Nous nous intéresserons dans le chapitre suivant aux limites et aux perspectives d'amélioration et d'extension de DVMS.

9

Perspectives autour de DVMS

Après nous être intéressés à la validation expérimentale de DVMS, nous détaillons dans ce chapitre les perspectives de travail autour de ce logiciel.

Nous proposons tout d'abord des compléments d'évaluation de DVMS ; nous évoquons également des moyens à mettre en œuvre pour corriger les limitations de l'approche ; nous voyons enfin comment étendre DVMS pour en faire un logiciel de gestion d'infrastructure virtualisée complet.

Sommaire

9.1 Compléments d'évaluation

L'étude expérimentale de DVMS qui a été menée au chapitre précédent peut être complétée de plusieurs manières, en évaluant la consommation en ressources du logiciel, en utilisant des traces d'exécution réelle et en comparant DVMS avec d'autres approches décentralisées.

9.1.1 Consommation des ressources

Il peut tout d'abord être intéressant de connaître la consommation en ressources de DVMS, pour s'assurer que celui-ci ne risque pas d'affecter le bon fonctionnement des machines virtuelles.

Nous savons déjà que DVMS utilise la totalité d'un cœur de processeur (et d'un seul) pendant le calcul d'un plan de reconfiguration. Étant donné que les nœuds de calcul récents disposent d'un grand nombre de cœurs, il serait cependant envisageable de dédier un cœur de chaque nœud de calcul à DVMS pour le faire fonctionner dans des conditions idéales, sans risquer de dégrader le fonctionnement des machines virtuelles.

Par ailleurs, nous savons que la consommation mémoire de DVMS augmente de manière plus ou moins importante lors du calcul d'un plan de reconfiguration, en fonction du nombre de nœuds et de machines virtuelles pris en compte : au plus quelques dizaines de mégaoctets avec Entropy 2 [HDL11] et au plus quelques centaines de mégaoctets avec Entropy 1 [HLM+09]. Néanmoins, nous avons vu au chapitre précédent que DVMS construisait presque toujours des partitions de très petite taille, ce qui fait que son empreinte mémoire est généralement négligeable.

La véritable inconnue réside dans le coût des communications réseau, notamment lorsque de nombreux événements sont transférés à un même nœud. Il serait bon de vérifier que la bande passante est faiblement utilisée en toutes circonstances et que les communications réseau ne sollicitent pas particulièrement le processeur, ce qui pourrait avoir une influence négative sur le passage à l'échelle de DVMS, voire dans le pire des cas affecter le bon fonctionnement des machines virtuelles. Ce point est cependant à relativiser, dans la mesure où les concepteurs des logiciels de gestion d'infrastructure virtualisée préconisent de mettre en place un réseau dédié au gestionnaire, qui est séparé du réseau auquel les machines virtuelles ont accès [VMw11, Apa12, Mic12].

9.1.2 Utilisation de traces d'exécution réelle

Un autre complément d'évaluation de DVMS réside dans l'utilisation de traces d'exécution réelle. Jusqu'à présent, nous avons en effet étudié DVMS en injectant une charge de travail aléatoire (une graine est utilisée pour assurer la reproductibilité des expériences) ; or si cela est utile pour s'assurer que DVMS se comporte correctement dans la plupart des cas, cela ne reflète en revanche pas le fonctionnement d'un système réel.

L'idée serait donc d'utiliser des traces d'exécution réelle pour injecter une charge de travail pertinente et vérifier que DVMS est capable de résoudre l'immense majorité des problèmes rapidement.

L'obtention de ces traces est cependant difficile, étant donné que la plupart des entreprises qui en disposent sont réticentes à les communiquer.

9.1.3 Comparaison avec d'autres approches décentralisées

Pour achever l'évaluation de DVMS, il serait bon de le comparer avec d'autres approches décentralisées pour l'ordonnancement dynamique de machines virtuelles.

Cependant, seul le code source de Snooze [FRM12] a été mis à disposition de la communauté, et les autres approches ne sont pas toujours suffisamment détaillées pour être mises en œuvre au travers d'un prototype logiciel.

9.2 Correction des limitations

Une fois l'évaluation de DVMS terminée, plusieurs possibilités s'offrent à nous pour venir à bout des limitations actuelles du prototype.

9.2.1 Gestion de la tolérance aux pannes

La première correction qui s'impose est de rendre DVMS tolérant aux pannes.

Réparation de l'anneau

Rendre DVMS tolérant aux pannes implique tout d'abord de rendre l'anneau tolérant aux pannes. À l'heure actuelle, si un nœud N_i tombe, un événement transitant par le nœud N_{i-1} ne peut plus arriver jusqu'au nœud N_{i+1}, à moins que N_{i-1} connaisse un raccourci vers un nœud N_j ($j \neq i$).

Pour préserver l'intégrité de l'anneau, nous pouvons par exemple utiliser les algorithmes proposés pour Chord [SMLN+03]. L'idée principale est de faire en sorte que chaque nœud connaisse non seulement son voisin, mais également son 2^1 ème successeur, son 2^2 ème successeur, et ainsi de suite jusqu'au 2^i ème successeur, i étant fixé par l'administrateur ; lorsqu'une partie de l'anneau tombe, la communication est toujours possible en passant par un successeur connu qui est encore fonctionnel.

Poursuite du traitement d'un événement

Rendre l'anneau tolérant aux pannes n'est cependant pas suffisant : il faut également rendre tout le processus de traitement d'un événement tolérant aux pannes.

Cela implique notamment que :

- le meneur d'une partition vérifie la bonne santé de l'initiateur avant de tenter de résoudre un événement ; si l'initiateur est défaillant, la partition peut être détruite, sous réserve qu'il n'y ait aucun problème sur un des autres nœuds de la partition ;
- l'initiateur d'un événement s'assure que le meneur est bien fonctionnel ; autrement, il faut relancer l'événement afin de poursuivre le traitement ;
- tout nœud défectueux soit exclus de la partition dont il faisait parti.

9.2.2 Amélioration de la gestion des événements

Outre la gestion de la tolérance aux pannes, il serait intéressant d'affiner la gestion des événements.

Construction pertinente des partitions

Afin d'améliorer la gestion des événements, il est tout d'abord envisageable de réviser la manière d'agrandir les partitions : plutôt que d'intégrer dans une partition le premier nœud libre rencontré, il faudrait sélectionner un nœud qui est susceptible de faire avancer la résolution du problème.

Une première approche pourrait consister à ce que les nœuds qui sont proches sur l'anneau s'échangent régulièrement des informations sur l'état de leurs ressources ; plutôt que de transférer un événement au premier nœud situé en dehors de la partition, il serait alors possible de sélectionner un nœud plus approprié en utilisant les informations d'état connues. Cette approche présente cependant deux inconvénients : (i) elle génère des communications réseau supplémentaires et (ii) les informations d'état peuvent devenir rapidement obsolètes, ce qui annule l'intérêt de l'approche.

Une autre possibilité serait de parcourir l'anneau normalement, mais de n'intégrer un nœud libre dans une partition que si l'état de ses ressources satisfait certaines conditions ; un exemple de condition serait, dans le cas d'un événement

« surutilisation », que le nœud visité puisse accueillir au moins une des machines virtuelles de l'initiateur.

Enfin, une dernière possibilité serait de fusionner/combiner les partitions qui sont associées à des événements complémentaires, par exemple un événement « surutilisation » et un événement « sous-utilisation ».

Limitation de la taille des partitions

Une fois les partitions construites de manière plus pertinente, il serait possible d'en limiter la taille afin de garantir la réactivité de DVMS ; nous avons en effet vu au chapitre précédent que quelques nœuds étaient suffisants dans la grande majorité des cas pour résoudre un événement.

La limitation de la taille des partitions ne doit cependant par être systématique ; elle est par exemple à éviter dans le cas d'un événement « surutilisation » car elle peut empêcher sa résolution ; en revanche, elle est particulièrement intéressante dans le cas d'un événement « sous-utilisation » pour éviter que la partition associée n'absorbe tous les nœuds libres de l'infrastructure.

Discrimination et priorité

La section précédente introduit la notion de discrimination entre (i) des événements de haute priorité, comme les événements liés à la surutilisation d'un nœud, qui doivent absolument être résolus pour éviter de pénaliser les utilisateurs, et (ii) des événements de basse priorité, comme les événements liés à la sous-utilisation d'un nœud, qui ne sont pas critiques dans la mesure où ils visent à optimiser l'utilisation de l'infrastructure.

Il est alors envisageable de faciliter l'agrandissement des partitions associées à des événements liés à la surutilisation d'un nœud. Par exemple, une partition bloquée associée à un événement « surutilisation » pourrait être autorisée à s'approprier les nœuds d'une partition en croissance liée à un événement « sous-utilisation ».

Traitement de nouveaux types d'événements

Une dernière amélioration de DVMS en matière de gestion des événements serait de le rendre capable de traiter de nouveaux types d'événements, comme l'arrivée de machines virtuelles dans le système ou encore la mise en maintenance d'un nœud.

L'arrivée d'une machine virtuelle pourrait être traitée comme un événement « surutilisation »: en considérant que la machine virtuelle est hébergée sur un nœud virtuel n'ayant aucune ressource, il faudrait alors démarrer (et non migrer) cette machine virtuelle sur un nœud réel ayant la capacité de l'accueillir.

La mise en maintenance d'un nœud pourrait également être mise en place au travers d'un événement « surutilisation »: il suffirait de spécifier que le nœud à mettre en maintenance ne dispose d'aucune ressource pour que ses machines virtuelles soient réparties sur d'autres nœuds.

9.2.3 Prise en compte des liens entre les machines virtuelles

Jusqu'à présent, nous avons toujours considéré les machines virtuelles de manière indépendante les unes des autres ; or, les algorithmes d'ordonnancement

d'Entropy permettent de gérer des groupes de machines virtuelles qui sont liées les unes aux autres, fonctionnalité qui a été perdue lors de l'intégration avec DVMS et qu'il serait intéressant de récupérer.

Spécification d'affinités et de répulsions

Les liens entre les machines virtuelles peuvent passer par la définition d'affinités et de répulsions (i) entre les machines virtuelles d'un même groupe, ou bien (ii) entre un groupe de machines virtuelles et un ensemble de nœuds.

Ces affinités et répulsions doivent être définies sur le nœud chargé de l'ordonnancement ; dans le cas de DVMS, cela implique que le meneur de chaque partition ait connaissance des contraintes de placement associées aux machines virtuelles dont il a la charge avant de débuter le calcul d'un plan de reconfiguration. Il est alors envisageable que chaque nœud connaisse les contraintes de placement relatives à chacune des machines virtuelles qu'il héberge et transmette ces informations au meneur sur demande de ce dernier.

Exécution d'une action sur un groupe de machines virtuelles

En dehors de la définition d'affinités et de répulsions, l'intérêt de définir des groupes de machines virtuelles réside dans la possibilité d'appliquer une même action sur l'ensemble des machines virtuelles d'un groupe, par exemple pour suspendre ou reprendre leur exécution, ou pour prendre un instantané.

La gestion des groupes de machines virtuelles pourrait être répartie entre les agents DVMS au moyen d'une table de hachage distribuée [RD01, RFH+01, MM02, SMLN+03, DHJ+07] ; notons que les actions effectuées sur un groupe de machines virtuelles ne doivent pas être incompatibles avec les migrations qu'un meneur est susceptible de réaliser, sous peine de générer des conflits d'ordonnancement.

9.3 Extension

Au-delà des corrections à apporter à DVMS, nous proposons plusieurs pistes de réflexion afin d'étendre ses fonctionnalités.

9.3.1 Gestion des images disque des machines virtuelles

Une première extension de DVMS serait liée à la gestion des images disque des machines virtuelles.

L'utilisation d'un système de stockage centralisé tel que NFS (*Network File System*) est à exclure, car il limiterait fortement le passage à l'échelle de DVMS.

En revanche, une architecture à deux niveaux, similaire à celle mise en œuvre avec OpenStack [Ope12], est envisageable : les partitions principales, non persistantes, des machines virtuelles seraient stockées localement (c'est-à-dire sur les nœuds hébergeant les machines virtuelles), tandis que les partitions utilisateur persistantes seraient réparties sur un ensemble de nœuds dédiés au stockage, de manière à répartir la charge d'entrées/sorties entre ces nœuds.

9.3.2 Gestion des infrastructures constituées de plusieurs centres de données et de calculs reliés par un réseau étendu

Une autre extension de DVMS consisterait à gérer des infrastructures constituées de plusieurs centres de données et de calculs reliés entre eux par un réseau étendu.

Dans ce type d'infrastructure, il faut notamment limiter les migrations de machines virtuelles d'un centre de données à un autre ; en effet, migrer une machine virtuelle sur un réseau étendu nécessite non seulement de transférer le contenu de sa mémoire, mais également de la totalité de son image disque, ce qui est d'autant plus coûteux que la bande passante du réseau étendu est réduite. Afin de limiter le coût d'une telle migration, lorsqu'elle s'avère nécessaire, il est possible de recourir à la déduplication [RMP12].

Deux possibilités s'offrent à nous pour limiter les migrations d'un centre de données à un autre :

1. intégrer la notion de bande passante entre chaque nœud et son voisin ; cela nécessite de disposer d'un ordonnanceur qui est capable de travailler avec cette information afin de privilégier les liens haut débit pour la migration des machines virtuelles ; un seul anneau est alors déployé sur l'ensemble des centres de données ;
2. un anneau spécifique est déployé sur chaque centre de données ; lorsqu'un événement ne peut pas être résolu sur un centre de données, l'initiateur fait alors appel à un médiateur qui se charge de le transférer à un autre centre de données.

9.3.3 Intégration dans un logiciel de gestion d'infrastructure virtualisée complet

La gestion des infrastructures constituées de plusieurs centres de données est un prélude à l'intégration de DVMS dans un logiciel de gestion d'infrastructure complet, capable de gérer notamment la connexion d'utilisateurs, les modèles de machines virtuelles, la soumission de machines virtuelles, et la collecte des informations d'état sur les machines virtuelles d'un utilisateur.

Cette dernière étape devrait être réalisée via l'initiative Discovery [LAGm12].

9.4 Conclusion

Dans ce chapitre, nous avons présenté plusieurs perspectives autour de DVMS.

Nous avons dans un premier temps détaillé les évaluations qui doivent être menées pour compléter la validation du prototype. Celles-ci touchent (i) à la consommation en ressources processeur, mémoire et réseau de DVMS, (ii) à l'utilisation de traces d'exécution réelle pour l'injection de charge dans les machines virtuelles, et (iii) à la comparaison avec d'autres approches décentralisées pour l'ordonnancement dynamique des machines virtuelles, lorsque cette comparaison est possible.

Par ailleurs, nous avons explicité plusieurs solutions pour corriger les limitations de DVMS concernant la tolérance aux pannes, la gestion des événements, et la prise en compte des liens entre les machines virtuelles comme le fait déjà Entropy.

Enfin, nous avons étudié les possibilités qui s'offraient à nous pour étendre les fonctionnalités de DVMS à la gestion (i) des images disque des machines virtuelles et (ii) des infrastructures constituées de plusieurs centres de données et de calculs reliés entre eux par un réseau étendu, le but final étant d'intégrer DVMS dans un logiciel de gestion d'infrastructure complet, logiciel qui est en cours d'élaboration dans le cadre de l'initiative Discovery [LAGm12].

Conclusion

Nous nous sommes intéressés dans cette thèse à la gestion des infrastructures distribuées d'un point de vue logiciel, et plus particulièrement à la conception d'une solution coopérative et décentralisée pour la gestion des infrastructures virtualisées.

Nous avons dans un premier temps présenté les principales catégories d'infrastructures distribuées existant de nos jours (notamment les grappes, les centres de données et de calculs, et les grilles) ainsi que les solutions logicielles traditionnellement utilisées pour les gérer (qu'elles se présentent sous la forme d'intergiciels [Fos06, LFF+06] ou de systèmes d'exploitation distribués [MvRT+90, PPD+95, LGV+05, Ril06, CFJ+08]).

Nous avons ensuite expliqué ce qu'est la virtualisation système [PG74, SN05] et ce qu'elle peut apporter aux fournisseurs et aux utilisateurs des infrastructures distribuées, notamment en matière de mutualisation des ressources, d'isolation entre utilisateurs, de facilité de configuration des environnements de travail et de maintenance de l'infrastructure. Nous avons remarqué à cette occasion que la virtualisation système avait contribué à l'essor des centrales numériques et d'un nouveau paradigme de calcul : l'informatique en nuage.

Puis nous avons étudié les fonctionnalités offertes par les principaux logiciels dédiés à la gestion des infrastructures virtualisées [NWG+09, SMLF09, VMw10, VMw11, Apa12, Cit12, Mic12, Ope12, nim13], notamment en matière de gestion du cycle de vie des machines virtuelles, de passage à l'échelle, de haute disponibilité et de tolérance aux pannes. Nous avons conclu cette étude en soulignant le caractère centralisé de ces logiciels de gestion, qui limite leur capacité à passer à l'échelle, dans un contexte où les infrastructures virtualisées comportent toujours plus de nœuds (jusqu'à plusieurs dizaines de milliers) [who13].

L'un des moyens d'améliorer le passage à l'échelle de ces logiciels de gestion est de décentraliser le traitement des tâches de gestion, lorsque cela s'avère judicieux. Or cette problématique de décentralisation a déjà été abordée par les recherches sur les systèmes d'exploitation distribués. C'est pourquoi nous nous sommes intéressés aux similarités entre les logiciels de gestion d'infrastructure virtualisée et les systèmes d'exploitation distribués [QL11] ; ces similarités existent aussi bien à l'échelle d'un nœud que de l'infrastructure tout entière, à ceci près que les entités manipulées par les logiciels de gestion d'infrastructure virtualisée ne sont plus des processus mais des machines virtuelles. Cette étude nous a permis d'identifier plusieurs contributions, notamment en matière de décentralisation de l'ordonnancement (dynamique) des machines virtuelles.

Avant de nous pencher sur cette décentralisation, nous avons tout d'abord insisté sur les limites d'une approche centralisée, en nous concentrant sur la problématique du passage à l'échelle : sur des infrastructures ayant une taille conséquente (ce qui, rappelons-le, est de plus en plus courant de nos jours), une telle approche ne permet pas de corriger rapidement les violations de qualité de service et peut se montrer suffisamment lente pour qu'un nouvel ordonnancement devienne caduc avant même d'avoir été entièrement appliqué. Nous avons ensuite réalisé un état de l'art des dernières contributions en matière d'ordonnancement décentralisé de machines virtuelles ; ces contributions suivent des approches hiérarchiques [FRM12] ou multi-agents [BDNDM10, YMF+10, MBP11, MMP11, RC11, FME12], qui permettent de répartir le travail d'ordonnancement entre plusieurs nœuds (voire tous les

nœuds) de l'infrastructure ; nous avons cependant observé à cette occasion que la quasi-totalité des approches demeurent partiellement centralisées.

Nous avons alors élaboré une nouvelle proposition pour ordonnancer les machines virtuelles de manière dynamique et décentralisée dans les centres de données et de calculs, proposition que nous avons baptisée DVMS (*Distributed Virtual Machine Scheduler*) [QL12, QLS12]. DVMS est déployé sous la forme d'un réseau d'agents organisés en anneau et qui coopèrent entre eux afin de traiter au plus vite les événements se produisant sur l'infrastructure, événements qui sont liés à la consommation en ressources des machines virtuelles hébergées par chaque nœud ; DVMS peut traiter plusieurs événements de manière simultanée et indépendante en partitionnant dynamiquement l'infrastructure, chaque partition ayant une taille appropriée à la complexité de l'événement qui lui est associé. Le parcours de l'anneau est optimisé afin d'éviter de traverser les nœuds qui sont déjà impliqués dans une partition. Par ailleurs, nous garantissons qu'un problème sera toujours résolu si une solution existe. Ces concepts ont été mis en œuvre dans un prototype écrit dans le langage de programmation Java. À l'heure actuelle, ce prototype tire parti des algorithmes d'ordonnancement conçus pour le gestionnaire de machines virtuelles Entropy [HLM+09, HDL11], qui permettent notamment :

- de restreindre le nombre de migrations, ce qui limite le coût de l'ordonnancement ;
- de gérer les dépendances séquentielles, c'est-à-dire le fait que la migration d'une machine virtuelle peut nécessiter de déplacer une autre machine virtuelle au préalable pour pouvoir être réalisée.

Cependant le prototype pourrait très bien exploiter d'autres algorithmes d'ordonnancement.

Nous avons ensuite décrit le protocole expérimental et les environnements de test qui ont permis de vérifier le bon fonctionnement du prototype. Nous nous sommes intéressés au paramétrage et au fonctionnement d'un logiciel conçu spécialement pour tester DVMS. Nous avons également exposé le déploiement de ce logiciel sur le banc de test Grid'5000 [gri13], afin d'évaluer DVMS au travers de simulations puis de mises en situation réelle ; notons que ces mises en situation réelle ont été facilitées par l'outil Flauncher [BCAC+12], qui a notamment permis d'automatiser la configuration des nœuds et des machines virtuelles, ainsi que l'injection de la charge de travail dans les machines virtuelles. Nous avons par ailleurs présenté l'environnement de simulation SimGrid [CLQ08] et souligné les avantages liés à son utilisation par rapport au déploiement du logiciel de test sur Grid'5000.

Nous avons analysé les résultats obtenus au travers des expériences réalisées avec les différents outils mentionnés précédemment. Ces résultats nous ont permis de comparer DVMS avec son équivalent centralisé Entropy, que ce soit pour la consolidation des machines virtuelles ou la correction de problèmes liés à la surutilisation des nœuds. Contrairement aux approches précédentes, DVMS s'est montré capable de gérer efficacement des infrastructures hébergeant des dizaines de milliers de machines virtuelles sur des milliers de nœuds. DVMS s'est bien comporté, en constituant des partitions de petite taille pour résoudre les événements ; ceci a permis à DVMS de se montrer beaucoup plus réactif qu'Entropy en résolvant chaque événement très rapidement. Par ailleurs, la réactivité de DVMS ne l'a pas empêché de soutenir la comparaison avec Entropy concernant le taux de consolidation des machines virtuelles.

Nous avons enfin envisagé plusieurs possibilités pour poursuivre les travaux autour de DVMS. Des expériences complémentaires peuvent être menées pour valider le prototype de manière plus approfondie, en évaluant sa consommation en ressources, en utilisant des traces d'exécution réelle et en comparant DVMS avec d'autres approches décentralisées lorsque les algorithmes sont disponibles. En outre, plusieurs corrections doivent être apportées à DVMS, surtout en matière de tolérance aux pannes et de gestion des événements. L'objectif final est d'intégrer DVMS dans un logiciel de gestion d'infrastructure virtualisée complet, logiciel qui est en cours d'élaboration dans le cadre de l'initiative Discovery [LAGm12].

Bibliographie

[Ada12] Adaptive Computing Enterprises, Inc. *TORQUE 4.0.2 Administrator Guide*, 2012. 31

[Adv08] Advanced Micro Devices, Inc., Sunnyvale, CA, USA. *AMD-V Nested Paging*, July 2008. 72

[AEW09] Andrea Arcangeli, Izik Eidus, and Chris Wright. Increasing memory density by using KSM. In *OLS '09: Proceedings of the Linux Symposium*, pages 19–28, July 2009. 41, 72

[And04] David P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Washington, DC, USA, November 2004. IEEE Computer Society. 28

[Apa12] Apache Software Foundation, Forest Hill, MD, USA. *Apache CloudStack 4.0.0-incubating Administrator's Guide*, May 2012. 17, 48, 62, 132, 139

[aws13] Amazon Web Services. http://aws.amazon.com/fr/, February 2013. 44

[azu13] Windows Azure. http://www.windowsazure.com, February 2013. 44

[BBHL08] Sukadev Bhattiprolu, Eric W. Biederman, Serge Hallyn, and Daniel Lezcano. Virtual servers and checkpoint/restart in mainstream Linux. *SIGOPS Oper. Syst. Rev.*, 42(5):104–113, July 2008. 37

[BCAC+12] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. Adding Virtualization Capabilities to Grid'5000. Technical Report RR-8026, INRIA, July 2012. 18, 113, 115, 140

[BCJ01] Rajkumar Buyya, Toni Cortes, and Hai Jin. Single System Image. *Int. J. High Perform. Comput. Appl.*, 15(2):124–135, May 2001. 32

[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, October 2003. ACM. 38, 39

[BDNDM10] Donato Barbagallo, Elisabetta Di Nitto, Daniel Dubois, and Raffaela Mirandola. A bio-inspired algorithm for energy optimization in a self-organizing data center. In Danny Weyns, Sam Malek, Rogério de Lemos, and Jesper Andersson, editors, *Self-Organizing Architectures*, volume 6090 of *Lecture Notes in Computer Science*, pages 127–151. Springer, Berlin/Heidelberg, Germany, 2010. 18, 83, 88, 89, 90, 139

[Bel05] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 41–46, Berkeley, CA, USA, April 2005. USENIX Association. 38, 39

[CBER09] Tim Cerling, Jeff Buller, Chuck Enstall, and Richard Ruiz. *Mastering Microsoft Virtualization*. Wiley Publishing, Inc., Indianapolis, Indiana, USA, December 2009. 36, 38, 39, 40

[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*, volume 2, pages 776–783, Washington, DC, USA, May 2005. IEEE Computer Society. 31, 112

[CFH+05] Christopher Clark, Keir Fraser, Steven Hand, Jacob G. 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 and Implementation*, pages 273–286, Berkeley, CA, USA, May 2005. USENIX Association. 41, 42, 74

[CFJ+08] Toni Cortes, Carsten Franke, Yvon Jégou, Thilo Kielmann, Domenico Laforenza, Brian Matthews, Christine Morin, Luis P. Prieto, and Alexander Reinefeld. XtreemOS: a Vision for a Grid Operating System. Technical report, XtreemOS, May 2008. 17, 28, 29, 31, 32, 139

[cfs13] CFS Scheduler. http://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt, February 2013. 69

[CH09] Matthew Chapman and Gernot Heiser. vNUMA: A virtual shared-memory multiprocessor. In *ATEC '09: Proceedings of the 2009 USENIX Annual Technical Conference*, pages 15–28. USENIX Association, June 2009. 38, 76

[Chi07] David Chisnall. *The Definitive Guide to the Xen Hypervisor*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007. 40, 70

[Cit12] Citrix Systems, Inc., Santa Clara, CA, USA. *Citrix XenServer 6.0 Administrator's Guide*, March 2012. 17, 36, 48, 62, 139

[CL85] K. Mani Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, February 1985. 74

[CLQ08] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *UKSIM '08: Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, pages 126–131, Washington, DC, USA, april 2008. IEEE Computer Society. 18, 107, 113, 115, 117, 130, 140

[Cre81] R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, 25(5):483 –490, sep. 1981. 35

[CWm+09] Horacio Andrés Lagar Cavilla, Joseph A. Whitney, Adin M. Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, and Mahadev Satyanarayanan. SnowFlock: rapid virtual machine cloning for cloud computing. In *EuroSys '09: Proceedings*

of the 4th ACM European conference on Computer systems, pages 1–12, New York, NY, USA, 2009. ACM. 73

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007. 135

[egi13] EGI - European Grid Infrastructure - towards a sustainable infrastructure. http://www.egi.eu/, February 2013. 27

[Eri09] Jonas Eriksson. Virtualization, Isolation and Emulation in a Linux Environment. Master's thesis, Umea University, SE-901 87 UMEA, SWEDEN, April 2009. 42

[Esk96] M. Rasit Eskicioglu. A comprehensive bibliography of distributed shared memory. *SIGOPS Oper. Syst. Rev.*, 30(1):71–96, January 1996. 76

[Euc12] Eucalyptus Systems, Inc., Goleta, CA, USA. *Eucalyptus 3.1.1 Administration Guide*, August 2012. 48

[FGNC01] G. Fedak, C. Germain, V. Neri, and F. Cappello. Xtremweb: a generic global computing system. In *CCGRID '01: Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 582 –587, may 2001. 28

[FM12] Eugen Feller and Christine Morin. Autonomous and energy-aware management of large-scale cloud infrastructures. In *IPDPSW '12: Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 2542 –2545, Washington, DC, USA, may 2012. IEEE Computer Society. 81, 88

[FME12] Eugen Feller, Christine Morin, and Armel Esnault. A Case for Fully Decentralized Dynamic VM Consolidation in Clouds. In *CloudCom '12: 4th IEEE International Conference on Cloud Computing Technology and Science*, Washington, DC, USA, December 2012. IEEE Computer Society. 18, 85, 87, 88, 89, 90, 139

[for13] Force.com. http://www.force.com/, February 2013. 44

[Fos06] Ian T. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*, 21(4):513–520, July 2006. 17, 31, 139

[fra13] France Grilles - Des solutions innovantes répondant à la croissance exponentielle des besoins de stockage et de traitement des données dans de nombreuses disciplines scientifiques. http://www.france-grilles.fr/, February 2013. 27

[FRM12] Eugen Feller, Louis Rilling, and Christine Morin. Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds. In *CCGRID '12: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 482–489, Washington, DC, USA, May 2012. IEEE Computer Society. 18, 75, 81, 88, 132, 139

[fut13] FutureGrid Portal. https://portal.futuregrid.org/, February 2013. 27

[FZRL08] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *GCE '08: Proceedings of Grid Computing Environments Workshop*, pages 1–10, Washington, DC, USA, November 2008. IEEE Computer Society. 26, 42, 44

[gae13] Google App Engine. https://developers.google.com/appengine/, February 2013. 44

[GHS02] A. Goscinski, M. Hobbs, and J. Silcock. GENESIS: an efficient, transparent and easy to use cluster operating system. *Parallel Comput.*, 28(4):557–606, April 2002. 73

[gri13] Grid'5000 - a scientific instrument designed to support experiment-driven research in all areas of computer science related to parallel, large-scale or distributed computing and networking. https://www.grid5000.fr, February 2013. 18, 27, 28, 29, 107, 112, 115, 117, 130, 140

[HDL11] Fabien Hermenier, Sophie Demassey, and Xavier Lorca. Bin repacking scheduling in virtualized datacenters. In *CP '11: Proceedings of the 17th international conference on Principles and practice of constraint programming*, pages 27–41, Berlin/Heidelberg, Germany, 2011. Springer. 75, 104, 105, 108, 117, 121, 130, 132, 140

[hem13] Hemera. https://www.grid5000.fr/mediawiki/index.php/Hemera, February 2013. 113

[HLM+09] Fabien Hermenier, Xavier Lorca, Jean M. Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 41–50, New York, NY, USA, March 2009. ACM. 42, 75, 104, 105, 108, 117, 130, 132, 140

[HLM10] Fabien Hermenier, Adrien Lèbre, and Jean-Marc Menaud. Cluster-Wide Context Switch of Virtualized Jobs. In *VTDC '10: Proceedings of the 4th International Workshop on Virtualization Technologies in Distributed Computing*, New York, NY, USA, June 2010. ACM. 42, 104, 105, 108

[HNIm11] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. Reactive consolidation of virtual machines enabled by postcopy live migration. In *VTDC '11: Proceedings of the 5th international workshop on Virtualization technologies in distributed computing*, pages 11–18, New York, NY, USA, June 2011. ACM. 74

[HUL06] Gernot Heiser, Volkmar Uhlig, and Joshua LeVasseur. Are virtual-machine monitors microkernels done right ? *SIGOPS Oper. Syst. Rev.*, 40(1):95–99, January 2006. 17, 67, 77

[HWF+05] Steven Hand, Andrew Warfield, Keir Fraser, Evangelos Kotsovinos, and Dan Magenheimer. Are virtual machine monitors microkernels done right ? In *HOTOS '05: Proceedings of the 10th conference on Hot Topics in Operating Systems*, volume 10, Berkeley, CA, USA, June 2005. USENIX Association. 17, 67, 77

[hyp13] Hyper-V – Configure Memory and Processors. http://technet.microsoft.com/en-us/library/cc742470.aspx, February 2013. 71

[kad13] Kadeploy – scalable, efficient and reliable deployment tool for clusters and Grid computing. http://kadeploy3.gforge.inria.fr/, February 2013. 112

[kav13] KaVLAN. https://www.grid5000.fr/mediawiki/index.php/KaVLAN, February 2013. 113

[KKL+07] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the Linux virtual machine monitor. In *OLS '07: Proceedings of the Linux Symposium*, volume 1, pages 225–230, June 2007. 36, 38, 39, 40, 70, 113

[Kot10] Evangelos Kotsovinos. Virtualization: Blessing or Curse ? *Queue*, 8(11):40–46, November 2010. 62

[LAGm12] Adrien Lèbre, Paolo Anedda, Massimo Gaggero, and Flavien Quesnel. Discovery, beyond the clouds. In *Euro-Par 2011: Parallel Processing Workshops*, volume 7156 of *Lecture Notes in Computer Science*, pages 446–456. Springer, Berlin/Heidelberg, Germany, 2012. 19, 136, 137, 141

[Leu04] Joseph Y. T. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Computer and Information Science. CRC Press LLC, Boca Raton, Florida, USA, 2004. 30

[LFF+06] E. Laure, S. M. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. Di Meglio, and A. Edlund. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006. 17, 28, 29, 31, 139

[LGV+05] R. Lottiaux, P. Gallard, G. Vallee, C. Morin, and B. Boissinot. OpenMosix, OpenSSI and Kerrighed: a comparative study. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid*, volume 2, pages 1016–1023, Washington, DC, USA, May 2005. IEEE Computer Society. 17, 31, 32, 75, 139

[lib13] Libvirt: The virtualization API. http://libvirt.org/, February 2013. 113

[Low09] Scott Lowe. *Introducing VMware vSphere 4*. Wiley Publishing Inc., Indianapolis, Indiana, first edition, September 2009. 42

[LPH+10] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, and R. Brightwell. Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing. In *IPDPS '10: Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium*, Washington, DC, USA, April 2010. IEEE Computer Society. 39, 40

[LSNG03] Benoît D. Ligneris, Stephen L. Scott, Thomas Naughton, and Neil Gorsuch. Open Source Cluster Application Resources (OSCAR): design, implementation and interest for the [computer] scientific community. In *HPCS '03: Proceedings of 17th Annual International*

Symposium on High Performance Computing Systems and Applications, May 2003. 73

[LY99] Tim Lindholm and Frank Yellin. *Java Virtual Machine Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1999. 37

[MBP11] Moreno Marzolla, Ozalp Babaoglu, and Fabio Panzieri. Server consolidation in Clouds through gossiping. In *WoWMoM '11: Proceedings of the 12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–6, Washington, DC, USA, June 2011. IEEE Computer Society. 18, 84, 87, 88, 89, 90, 139

[MDP+00] Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. *ACM Comput. Surv.*, 32(3):241–299, September 2000. 74

[Mic12] Microsoft Corporation, Redmond, WA, USA. *System Center 2012 – Virtual Machine Manager Technical Documentation*, April 2012. 17, 48, 62, 75, 132, 139

[MJL13] Paul Menage, Paul Jackson, and Christoph Lameter. cgroups. http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt, February 2013. 40

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer, Berlin/Heidelberg, Germany, 2002. 135

[MMP11] Carlo Mastroianni, Michela Meo, and Giuseppe Papuzzo. Self-economy in cloud data centers: statistical assignment and migration of virtual machines. In *Euro-Par '11: Proceedings of the 17th international conference on Parallel processing*, volume 1, Berlin/Heidelberg, Germany, 2011. Springer. 18, 75, 85, 88, 89, 90, 139

[MvRT+90] S. J. Mullender, G. van Rossum, A. S. Tananbaum, R. van Renesse, and H. van Staveren. Amoeba: a distributed operating system for the 1990s. *Computer*, 23(5):44–53, May 1990. 17, 31, 32, 139

[NAMG09] Lucas Nussbaum, Fabienne Anhalt, Olivier Mornard, and Jean-Patrick Gelas. Linux-based virtualization for HPC clusters. In *OLS '09: Proceedings of the Linux Symposium*, pages 221–234, July 2009. 40

[nim13] Nimbus 2.10. http://www.nimbusproject.org/docs/2.10/, February 2013. 17, 48, 62, 139

[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 *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, volume 0, pages 124–131, Washington, DC, USA, May 2009. IEEE Computer Society. 17, 48, 62, 139

[Ope12] OpenStack, LLC, San Antonio, TX, USA. *OpenStack Compute Administration Manual*, folsom edition, November 2012. 17, 48, 62, 135, 139

[ope13] OpenNebula 3.8. http://opennebula.org/documentation:rel3.8, February 2013. 48

[osg13] The Open Science Grid. https://www.opensciencegrid.org/bin/view, February 2013. 27

[paj13] Pajé Visualization Tool – analysis of execution traces. http://paje.sourceforge.net/, February 2013. 114

[PEL11] Eunbyung Park, Bernhard Egger, and Jaejin Lee. Fast and space-efficient virtual machine checkpointing. In *VEE '11: Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, New York, NY, USA, March 2011. ACM. 41

[PG74] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974. 17, 35, 39, 139

[PLS13] Martin Prpič, Rüdiger Landmann, and Douglas Silas. Red Hat Enterprise Linux 6 Resource Management Guide. https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Resource_Management_Guide/index.html, February 2013. 40

[PPD+95] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8:221–254, 1995. 17, 31, 32, 139

[QL11] Flavien Quesnel and Adrien Lèbre. Operating Systems and Virtualization Frameworks: From Local to Distributed Similarities. In *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. 17, 67, 139

[QL12] Flavien Quesnel and Adrien Lèbre. Cooperative Dynamic Scheduling of Virtual Machines in Distributed Systems. In *Euro-Par 2011: Parallel Processing Workshops*, volume 7156 of *Lecture Notes in Computer Science*, pages 457–466. Springer, Berlin/Heidelberg, Germany, 2012. 19, 93, 95, 104, 140

[QLS12] Flavien Quesnel, Adrien Lèbre, and Mario Südholt. Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS. *Concurrency and Computation: Practice and Experience*, 2012. 19, 93, 97, 104, 140

[RC11] Jonathan Rouzaud Cornabas. A Distributed and Collaborative Dynamic Load Balancer for Virtual Machine. In *Euro-Par 2010 Parallel Processing Workshops*, volume 6586 of *Lecture Notes in Computer Science*, pages 641–648. Springer, Berlin/Heidelberg, Germany, August 2011. 18, 75, 86, 88, 89, 90, 139

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, Berlin/Heidelberg, Germany, 2001. 135

[REH07] Timothy Roscoe, Kevin Elphinstone, and Gernot Heiser. Hype and virtue. In *HOTOS '07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, pages 1–6, Berkeley, CA, USA, May 2007. USENIX Association. 17, 67, 77

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, August 2001. ACM. 135

[RI00] John S. Robin and Cynthia E. Irvine. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In *SSYM '00: Proceedings of the 9th conference on USENIX Security Symposium*, pages 129–144, Berkeley, CA, USA, August 2000. USENIX Association. 39

[Ril06] Louis Rilling. Vigne: Towards a Self-Healing Grid Operating System. In *Proceedings of Euro-Par 2006*, volume 4128 of *Lecture Notes in Computer Science*, pages 437–447, Berlin/Heidelberg, Germany, August 2006. Springer. 17, 31, 32, 139

[RMP12] Pierre Riteau, Chritine Morin, and Thierry Priol. Shrinker: efficient live migration of virtual clusters over wide area networks. *Concurrency and Computation: Practice and Experience*, 2012. 136

[Rot94] H. G. Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. *IEE Proceedings - Computers and Digital Techniques*, 141(1):1–10, January 1994. 30, 80

[Rus07] Rusty Russel. lguest: Implementing the little Linux hypervisor. In *OLS '07: Proceedings of the Linux Symposium*, volume 2, pages 173–178, June 2007. 39

[sal13] Salesforce.com – CRM and cloud computing. http://www.salesforce.com/, February 2013. 44

[sch13] Linux scheduling domains. http://www.kernel.org/doc/Documentation/scheduler/sched-domains.txt, February 2013. 70

[set13] SETI@home. http://setiathome.berkeley.edu/, February 2013. 28

[SG98] Abraham Silberschatz and Peter B. Galvin. *Operating System Concepts*. Addison-Wesley, Reading, Massachusetts, USA, 5th edition, August 1998. 13, 69

[sim13] SimGrid: versatile simulation of distributed systems. http://simgrid.gforge.inria.fr/, February 2013. 113

[SK10] Udo Steinberg and Bernhard Kauer. NOVA: a microhypervisor-based secure virtualization architecture. In *EuroSys '10: Proceedings of the 5th European conference on Computer systems*, pages 209–222, New York, NY, USA, April 2010. ACM. 39, 40

[SMLF09] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5):14–22, September 2009. 17, 48, 62, 139

[SMLN+03] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003. 133, 135

[SN05] James E. Smith and Ravi Nair. *Virtual machines: versatile platforms for systems and processes*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005. 17, 35, 36, 72, 139

[SPF+07] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, volume 41, pages 275–287, New York, NY, USA, March 2007. ACM. 37

[Sta08] William Stallings. *Operating Systems Internals and Design Principles*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, sixth edition, July 2008. 30

[tak13] Taktuk – Adaptive large scale remote executions deployment. http://taktuk.gforge.inria.fr/, February 2013. 112

[Tan01] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, second edition, March 2001. 30, 71, 72, 75

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurr. Comput. : Pract. Exper.*, 17:323–356, February 2005. 31

[UNR+05] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C. M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kagi, Felix H. Leung, and Larry Smith. Intel virtualization technology. *Computer*, 38(5):48–56, May 2005. 39

[VMw09] VMware, inc., Palo Alto, CA, USA. *VMware vSphere 4: The CPU Scheduler in VMware ESX 4*, 2009. 70

[VMw10] VMware, Inc., Palo Alto, CA, USA. *VMware vCloud Architecting a vCloud*, 2010. 17, 48, 62, 139

[VMw11] VMware, Inc., Palo Alto, CA, USA. *VMware vSphere Basics*, 2009-2011. 17, 48, 62, 75, 132, 139

[Vog08] Werner Vogels. Beyond Server Consolidation. *Queue*, 6(1):20–26, 2008. 42

[vsm13] Versatile SMP (vSMP) Architecture. http://www.scalemp.com/products/, February 2013. 76

[Wal02] Carl A. Waldspurger. Memory resource management in VMware ESX server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, December 2002. 36, 38, 41, 72

[who13] Who Has the Most Web Servers ? http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-most-web-servers/, February 2013. 17, 81, 139

[win13] Windows Scheduling. http://msdn.microsoft.com/en-us/library/windows/desktop/ms685096.aspx, February 2013. 69

[wlc13] WLCG - Worldwide LHC Computing Grid. http://wlcg.web.cern.ch/, February 2013. 27

[WLS+09] Timothy Wood, Gabriel T. Levin, Prashant Shenoy, Peter Desnoyers, Emmanuel Cecchet, and Mark D. Corner. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. *SIGOPS Oper. Syst. Rev.*, 43(3):27–36, July 2009. 76

[xse13] XSEDE - Extreme Science and Engineering Discovery Environment. https://www.xsede.org/home, February 2013. 27

[YMF+10] Yagiz O. Yazir, Chris Matthews, Roozbeh Farahbod, Stephen Neville, Adel Guitouni, Sudhakar Ganti, and Yvonne Coady. Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In *Cloud '10: IEEE 3rd International Conference on Cloud Computing*, pages 91–98, Los Alamitos, CA, USA, July 2010. IEEE Computer Society. 18, 75, 84, 88, 89, 90, 139

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

ECOLE DES MINES DE NANTES

Thèse de Doctorat

Flavien QUESNEL

Vers une gestion coopérative des infrastructures virtualisées à large échelle : le cas de l'ordonnancement

Toward Cooperative Management of Large-scale Virtualized Infrastructures: the Case of Scheduling

Résumé

Les besoins croissants en puissance de calcul sont généralement satisfaits en fédérant de plus en plus d'ordinateurs (ou nœuds) pour former des infrastructures distribuées.
La tendance actuelle est d'utiliser la virtualisation système dans ces infrastructures, afin de découpler les logiciels des nœuds sous-jacents en les encapsulant dans des machines virtuelles. Pour gérer efficacement ces infrastructures virtualisées, de nouveaux gestionnaires logiciels ont été mis en place.
Ces gestionnaires sont pour la plupart hautement centralisés (les tâches de gestion sont effectuées par un nombre restreint de nœuds dédiés). Cela limite leur capacité à passer à l'échelle, autrement dit à gérer de manière réactive des infrastructures de grande taille, qui sont de plus en plus courantes.
Au cours de cette thèse, nous nous sommes intéressés aux façons d'améliorer cet aspect ; l'une d'entre elles consiste à décentraliser le traitement des tâches de gestion, lorsque cela s'avère judicieux.
Notre réflexion s'est concentrée plus particulièrement sur l'ordonnancement dynamique des machines virtuelles, pour donner naissance à la proposition DVMS (*Distributed Virtual Machine Scheduler*).
Nous avons mis en œuvre un prototype, que nous avons validé au travers de simulations (notamment via l'outil SimGrid), et d'expériences sur le banc de test Grid'5000. Nous avons pu constater que DVMS se montrait particulièrement réactif pour gérer des infrastructures virtualisées constituées de dizaines de milliers de machines virtuelles réparties sur des milliers de nœuds.
Nous nous sommes ensuite penchés sur les perspectives d'extension et d'amélioration de DVMS. L'objectif est de disposer à terme d'un gestionnaire décentralisé complet, objectif qui devrait être atteint au travers de l'initiative Discovery qui fait suite à ces travaux.

Mots clés

virtualisation, infrastructures distribuées, ordonnancement dynamique, systèmes multi-agents, gestion événementielle, systèmes coopératifs, systèmes autonomes, passage à l'échelle, réactivité.

Abstract

The increasing need in computing power has been satisfied by federating more and more computers (called nodes) to build the so-called distributed infrastructures.
Over the past few years, system virtualization has been introduced in these infrastructures (the software is decoupled from the hardware by packaging it in virtual machines), which has lead to the development of software managers in charge of operating these virtualized infrastructures.
Most of these managers are highly centralized (management tasks are performed by a restricted set of dedicated nodes). As established, this restricts the scalability of managers, in other words their ability to be reactive to manage large-scale infrastructures, that are more and more common.
During this Ph.D., we studied how to mitigate these concerns; one solution is to decentralize the processing of management tasks, when appropriate.
Our work focused in particular on the dynamic scheduling of virtual machines, resulting in the DVMS (Distributed Virtual Machine Scheduler) proposal.
We implemented a prototype, that was validated by means of simulations (especially with the SimGrid tool) and with experiments on the Grid'5000 testbed. We observed that DVMS was very reactive to schedule tens of thousands of virtual machines distributed over thousands of nodes.
We then took an interest in the perspectives to improve and extend DVMS. The final goal is to build a full decentralized manager. This goal should be reached by the Discovery initiative, that will leverage this work.

Key Words

virtualization, distributed infrastructures, dynamic scheduling, multi-agent systems, event-based systems, cooperative systems, autonomous systems, scalability, reactivity.