HAL
open science

Environnement d'exécution pour des services de calcul à la demande sur des grappes mutualisées

Rodrigue Chakode Noumowe Chakode Noumowe

To cite this version:

Rodrigue Chakode Noumowe Chakode Noumowe. Environnement d'exécution pour des services de calcul à la demande sur des grappes mutualisées. Autre [cs.OH]. Université de Grenoble, 2012. Français. NNT : 2012GRENM035 . tel-00744409

HAL Id: tel-00744409

https://theses.hal.science/tel-00744409v1

Submitted on 23 Oct 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 2012/06/26

Présentée par

Rodrigue Chakode

Thèse dirigée par **Jean-François Méhaut**
et codirigée par **Maurice Tchuenté**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
et de l'**Ecole doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique, Grenoble Universités**

Environnement d'Exécution pour des Services de Calcul à la Demande sur des Grappes Mutualisées

Thèse soutenue publiquement le **26/06/2012**,
devant le jury composé de :

Mme. Christine Collet
Professeur, Grenoble INP - Ensimag, Présidente
M. Frédéric Desprez
Directeur de Recherche, INRIA Lyon, Rapporteur
M. Daniel Hagimont
Professeur, INPT/ENSEEIHT Toulouse, Rapporteur
M. Mathias Silvant
CEO EdXact Voiron, Examinateur
M. Jean-François Méhaut
Professeur, Université de Grenoble, Directeur de thèse
M. Maurice Tchuenté
Professeur, Université de Yaoundé I, Co-Directeur de thèse

"Suis le chemin qui est le tien, et force toi de travailler pour obtenir ce que tu désires." (Tchieudjie Ngaha aka Mâ Lou, ma chère maman à quije dédie cette thèse pour son amour et son dévouement pour mon éducation)

Remerciements

Presque quatre années se sont écoulées, c'est terminé. J'ai enfin le titre de Docteur ou plus simplement "Docta" comme on dit chez moi au Cameroun. La joie se trouve au bout de la difficulté, disait ma grand-mère. Avant de clore cette aventure, je souhaite exprimer ici ma gratitude à tous ceux et celles qui ont contribués à son accomplissement.

D'abord je voudrais remercier Jean-François Méhaut, mon directeur de thèse, pour m'avoir accepté comme thésard et pour son encadrement. Jean-François, tu n'as pas été qu'un directeur de thèse. Que ce soient pour des démarches personnelles ou pour des démarches administratives qui ont été ô combien fréquentes et lourdes, ta disponibilité et ton soutien ont été sans faille. Je t'en remercie infiniment et espère que tu as trouvé en notre collaboration les fruits escomptés.

Merci au Professeur Maurice Tchuenté, mon co-directeur de thèse, qui m'a également initié à la recherche depuis mon DEA. Merci pour ta disponibilité et l'opportunité que tu m'as offerte en me présentant à Jean-François Méhaut et à Brigitte Plateau, ce qui a été le point de départ de cette thèse.

Merci aux membres du jury qui ont bien voulus évaluer mes travaux : Fréderic Desprez et Daniel Hagimont qui ont acceptés la lourde tâche de rapporteur, les examinateurs Christine Collet qui a également présidée la soutenance et Mathias Silvant que je voudrais par ailleurs remercier pour notre collaboration fructueuse durant mes travaux de recherche.

Cette thèse n'aurait pas eu lieu sans le financement du pôle de compétitivité Minalogic à travers le projet Ciloe. Je remercie l'INRIA qui m'a recruté dans le cadre de ce projet.

Merci au Laboratoire d'Informatique de Grenoble qui m'a accueilli, particulièrement à Brigitte Plateau pour la collaboration qu'elle a contribuée à mettre en place avec le Cameroun. Merci à tous les membres des équipes Mescal et Moais, spécialement à Ahlem, Annie Claude, Annie Simon, Christian Seguy pour leur constante disponibilité ; à Vania pour ses encouragements et son attention ; à Christiane Pousa – ma fabuleuse collègue de bureau – pour ses conseils, son amitié et les confiseries suisses :) ; et à mes compagnons de tous les jours Marcio, Laurent, Lucas, Christophe, Augustin, Vinicius, Kiril, Christian, Emilio...

Merci à Alexandre Carissimi et Blaise Yenke pour votre disponibilité, votre sympathie infinie, vos encouragements et la collaboration que nous avons eue.

J'ai une pensée particulière pour mes amis de Grenoble : Eric, Priscille, Nora, Florence, Léonie, Linda, Christiane (mbôbô), Alexandre, Kevin, Marcel, Leon, Serge, Steeve, Sandra et tous les autres. Mon séjour aurait été sans doute moins agréable si je ne vous avais pas rencontrés. Une fois de plus merci Sandra pour ton soutien, tes relectures et toute l'amour que tu m'as porté et me porte encore. Merci d'avoir accepté de partager ma vie.

Sans être exhaustif, je voudrais remercier toutes ces personnes que j'ai pour la plupart rencontrées durant mon cursus et qui me sont aujourd'hui très proches : Simplice, Ronald, Christiane (CK), Innocent, Donatien, Yanik, Désiré, Agnès, Larissa, Dieudonné, Alain...

Mille merci à ma famille pour son soutien et mon éducation depuis ma tendre enfance : ma mère Tchieudjie Ngaha, mes tantes Bibiane Welawa et Suzanne Nouné, mes frères, cousins et cousines Michel Tchakoundé, Robert Tsangué, Théophile Tchitcha, Elysée Bogning...

Faute de place je ne peux malheureusement citer tout le monde ici et m'en excuse.

Résumé

Cette thèse étudie la gestion de ressources pour des services de calcul intensif à la demande sur une grappe de calcul partagée. L'objectif étant de définir des outils d'exploitation qui permettent d'allouer dynamiquement les ressources pour l'exécution des requêtes à la demande, de partager équitablement les ressources entre les différents services, tout en maximisant leur utilisation. Financé par le pôle de compétitivité Minalogic [1], ce travail s'adresse à des petites organisations de types PME ou PMI dont les budgets de fonctionnement ne permettent pas de supporter les charges d'une infrastructure de calcul dédiée.

La première partie présente un état de l'art sur la gestion de ressources dans les domaines de nuage de calcul et de calcul intensif. Puis, tirant partie de cette étude, nous avons défini et mis en œuvre une architecture virtualisée pour faciliter l'exécution dynamique des requêtes en s'appuyant sur un gestionnaire de ressources spécifique que nous avons développé. Nous avons enfin proposé une stratégie et des algorithmes d'ordonnancement de tâches qui permettent de partager proportionnellement les ressources entre les services des différentes entreprises tout en maximisant l'utilisation. En effet, cette stratégie qui repose sur des techniques de bail de ressources introduit une flexibilité qui offre un compromis efficace entre équité et une meilleure utilisation de ressources.

Pour évaluer notre travail, nos contributions concernant notamment la gestion dynamique de tâches et les algorithmes de partage de ressources ont été implémentées dans un prototype logiciel que nous avons développé. Basé sur des standards ouverts, ce prototype s'appuie sur des outils de virtualisation ouverts et performants tels que OpenNebula et Xen pour allouer et manipuler les machines virtuelles sur les nœuds de la grappe. Notre architecture a été évaluée à partir de ce prototype et diverses charges de travail qui ont été injectées sur une plateforme expérimentale en grandeur nature déployée dans Grid'5000. Les résultats montrent que ces différentes contributions satisfont les objectifs fixés tout en étant performantes et efficaces.

Mot clés : Calcul à la demande, Gestion de ressources, Virtualisation, Environnement d'exécution dynamique, Calcul haute performance.

1. Financement dans le cadre du projet Ciloe (http ://ciloe.minalogic.net).

Abstract

This thesis studies resource management for on-demand computing services through a shared cluster. In such a context, the aim was to propose tools to enable allocating resources automatically for executing on-demand user requests, to enable sharing resources proportionally among those services, while maximizing their use. Funded by the Minalogic [2] global business cluster, this work targets on organizations such as SMB, which are not able to support the charge of purchasing and maintaining a dedicated computing infrastructure.

Firstly, we have achieved a deep survey around on-demand computing and resource management techniques in the areas of high performance computing and cloud computing. According to this survey, we have defined a virtualized architecture to enable dynamic execution of user requests thanks to a special resource manager. Finally, we have proposed policies and algorithms that allow to share the resources among the services of the different businesses while maximizing the utilization. Indeed, this policy relies on a flexibility introduced, thanks to principles of resource leasing, in our algorithms to make a suitable tradeoff between equity and utilization.

For evaluating our work, our contributions notably for dealing dynamically with resources allocation as well as the algorithms for scheduling tasks have been implemented in a software prototype that we have developed. Based on open standards, this prototype relies on existing virtualization tools such as OpenNebula for allocating and manipulating virtual machines over the cluster's nodes. From this prototype along with various workloads, we have carried out experiments to evaluate our architecture and scheduling algorithms. Results have shown that our contributions allow to achieve the expected goals while being reliable and efficient.

Key-words : On-demand computing, Resource management, Virtualization, Dynamic execution environment, High Performance Computing.

2. This work was carried out in the context of the Ciloe Project (http ://ciloe.minalogic.net).

Table des matières

Résumé **1**

Abstract **2**

1 Introduction **9**
1.1 Contexte industriel 10
1.2 Objectifs et contributions 10
1.2.1 Objectifs 11
1.2.2 Contributions 11
1.3 Organisation du document 13

2 Calcul à grande échelle : des systèmes parallèles aux nuages de calcul **15**
2.1 Contextes des plateformes de calcul 16
2.1.1 Calcul haute performance 16
2.1.2 Calcul dans le nuage 17
2.1.2.1 Concepts 18
2.1.2.2 Modèles de services 19
2.1.2.3 Types de nuages 21
2.2 Organisations matérielles d'une plateforme de calcul 22
2.2.1 Grappes d'ordinateurs 22
2.2.2 Grilles 23
2.2.3 Systèmes pair-à-pair 24
2.3 Exploitation d'une plateforme de calcul 25
2.3.1 Optimisation et parallélisation des applications 25
2.3.2 Ordonnancement de tâches et gestion de ressources 26
2.3.2.1 Politiques d'ordonnancement 27
2.3.2.2 Placement de tâches sur les ressources 30
2.3.3 Virtualisation de ressources 31
2.3.3.1 Comprendre la virtualisation 31
2.3.3.2 Techniques de virtualisation 32
2.3.3.3 Virtualisation d'une infrastructure distribuée 35
2.3.3.4 Étude de technologies de virtualisation 36
2.3.3.5 Les plus et les moins de la virtualisation 42
2.4 Synthèse 43

3 De l'intérêt d'une mutualisation de ressources aux problématiques d'exploitation 45
3.1 Pourquoi la mutualisation de ressources ? 45
3.2 La mutualisation et ses contraintes de gestion de ressources 46
3.2.1 Principe de la mutualisation 47
3.2.2 Cas d'étude et objectifs 47
3.3 A propos de l'existant 49
3.3.1 Positionnement dans l'écosystème des nuages de calcul 49
3.3.2 Mécanismes de partage de ressources 49
3.3.2.1 Partage par découpage statique 50
3.3.2.2 Partage équitable ou *Fair-share* 51
3.3.2.3 Améliorations du partage équitable : préemptions/migrations de tâches 52
3.3.3 Environnement d'exécution de tâches 53
3.3.3.1 Environnement d'exécution sur une machine réelle 53
3.3.3.2 Environnement d'exécution au sein d'une machine réelle . . 54
3.4 Synthèse : de la gestion de ressources mutualisées 54

4 Mutualisation d'une plateforme pour des services de calcul à la demande 57
4.1 Présentation du problème 57
4.2 Vers une plateforme mutualisée pour des services de calcul 58
4.3 Virtualiser pour favoriser la dynamicité et l'automatisation 58
4.4 Allocation et gestion flexibles de baux de ressources virtuelles 60
4.4.1 Types de tâches : tâches de service et tâches best-effort 61
4.4.2 Baux de machines virtuelles : bail préemptif et bail non-préemptif . . 61
4.4.3 Un mécanisme de partage de ressources flexible 62
4.5 Discussions et synthèse 64

5 Plateforme logicielle pour la mutualisation de ressources virtualisées pour des services de calcul 67
5.1 Architecture 67
5.1.1 Architecture logicielle 67
5.1.1.1 Gestionnaire SaaS 68
5.1.1.2 Virtualisation de la grappe sur un socle OpenNebula 69
5.1.2 Architecture de déploiement 69
5.2 Fonctionnement de la plateforme : Focus sur la gestion de ressources 71
5.2.1 Admission de requêtes 72
5.2.2 Sélection de tâches 73
5.2.2.1 Tâches de service 73
5.2.2.2 Tâches best-effort 75
5.2.3 Exécution de tâche : Création transparente de machine virtuelle 76
5.2.4 Contrôle des exécutions et supervision de la plateforme 77
5.2.5 Cycle de vie d'une tâche 77
5.3 Synthèse 79

6 SVMSched : un prototype du gestionnaire de services **81**
6.1 Composantes architecturales 81
6.1.1 Point d'accès 81
6.1.2 Gestionnaire de mutualisation 82
6.1.3 Tour de contrôle 84
6.2 Détails d'implémentation 85
6.2.1 Configuration de SVMSched et définition des règles de mutualisation 85
6.2.2 Création et déploiement transparents d'une machine virtuelle 87
6.3 Synthèse 88

7 Expérimentation et évaluation **89**
7.1 Conditions expérimentales 89
7.1.1 Environnement logiciel et applications 89
7.1.2 Environnement matériel et déploiement 90
7.1.3 Charge de travail 91
7.2 Expériences et analyse des résultats 92
7.2.1 Performances de la virtualisation 92
7.2.1.1 Temps de démarrage d'une tâche 92
7.2.1.2 Surcouts de la virtualisation 93
7.2.2 Analyse du partage de ressources 94
7.2.2.1 Partage statique, partage flexible : impact sur l'utilisation . . 95
7.2.2.2 Intérêt du dimensionnement 96
7.2.3 Passage à l'échelle 97
7.3 Discussions 98

8 Conclusion et Perspectives **101**
8.1 Contributions 101
8.2 Perspectives 102
8.3 Conclusions et commentaires personnels 103

Annexe **109**

Index **109**

Références et Bibliographie **113**

Table des figures

2.1 Vue générale d'une plateforme de calcul. 15
2.2 Exemple d'un problème nécessitant des calcul 17
2.3 Aperçu d'une plateforme de calcul dans le nuage. 19
2.4 Grappe d'ordinateurs 23
2.5 Grappe de machines virtuelles 23
2.6 Grille de calcul 24
2.7 Illustration d'un système de pair-à-pair. 25
2.8 Politique premier-arrivé, premier-servi. 27
2.9 Politique premier-arrivé, premier-servi avec remplissage. 28
2.10 Principe de la virtualisation et de la consolidation 31
2.11 Principe de la virtualisation complète sur une architecture $x86$. 33
2.12 Principe de la paravirtualisation sur une architecture $x86$. 33
2.13 Principe de la virtualisation assistée par le matériel sur une architecture $x86$. 34

3.1 Charge de Grid'5000 entre mi-novembre et mi-décembre 2011. 46
3.2 Hébergement multitenant. 47
3.3 Plateforme de nuage avec des ressources partagées 48
3.4 Partage de ressources par découpage statique. 50
3.5 Découpage statique vs Partage équitable. 52

4.1 Vision abstraite de l'architecture logicielle. 59
4.2 Illustration du mécanisme d'allocation de ressources. 63

5.1 Architecture logicielle. 68
5.2 Architecture physique. 70
5.3 Système de queues 72
5.4 Cycle de vie d'une tâche 78

6.1 Architecture et flot de traitement de SVMSched. 82
6.2 Extrait d'un journal d'événements de SVMSched. 83
6.3 Extrait d'un journal de supervision de SVMSched. 84
6.4 Exemple de configuration de SVMSched. 86

7.1 Temps de création d'une machine virtuelle (DomU) 93
7.2 Évaluation des surcouts de performances de la virtualisation 95
7.3 Impact du mécanisme de partage de ressources sur l'utilisation et le temps global d'exécution 96
7.4 Utilisation de ressources lorsque les proportions tiennent compte des besoins. 97

7.5 Exécution avec une charge de travail de 1000 tâches sur une plateforme de $20*8 = 160$ cœurs. 97

Introduction 1

Cette thèse propose des méthodologies et mécanismes pour gérer des ressources dans un nuage de services de calcul intensif reposant sur une plateforme [1] de calcul partagée.

En effet, avec l'émergence du calcul dans le nuage (ou *Cloud Computing* en anglais), les traitements informatiques traditionnellement exécutés sur le poste utilisateur sont déportés sur des infrastructures distantes. Un des intérêts est que les utilisateurs n'ont plus à installer les applications sur leurs postes de travail. Ainsi ce n'est plus à eux d'assurer les mises à jour de ces applications. Elles sont hébergées chez les éditeurs et accessibles à distance comme des services à la demande. C'est le modèle *logiciel en tant que service* plus connu sous l'anglicisme *Software-as-a-Service*, SaaS. Pour l'éditeur de logiciel, cela permet d'être plus compétitif en élargissant son marché. Notamment, contrairement à la méthode classique de distribution basée sur des licences, son logiciel sera plus accessible pour des clients ayant des moyens d'investissement limités ou ceux ayant des besoins ponctuels d'utilisation.

Pour mettre en œuvre un nuage SaaS, des nuages de type PaaS (*Plateform-as-a-Service*) sont disponibles. Typiquement commercial ou gratuit en version limitée, un nuage PaaS offre une plateforme intégrée incluant à la fois des ressources matérielles et des outils logiciels pour développer, déployer et héberger des services. Présentant l'avantage d'être des solutions clé-en-main qui réduisent le temps de mise en œuvre de SaaS, elles sont cependant particulièrement destinées à des applications web et présentent pour cela des limitations pour des applications de calcul intensif.

Une autre alternative consiste à déployer les services par le biais d'une infrastructure matérielle propre à l'entreprise. Les ressources de l'infrastructure doivent fournir une capacité de calcul suffisante pour supporter le service, avec de potentielles fluctuations et baisses soudaines de la demande. Cela exige donc de disposer d'une plateforme de calcul appropriée. Cependant, une plateforme de calcul reste onéreuse au point où disposer une telle plateforme est généralement hors de portée pour une entreprise ou organisation de petite ou de moyenne taille (PME ou PMI). Il est en effet difficile pour une telle entreprise, dont les capacités d'investissement sont limitées, de supporter les charges liées à l'acquisition, l'exploitation et le fonctionnement de l'infrastructure : nous pensons, par exemple, aux couts des serveurs, des interconnexions réseaux, du système de refroidissement et d'un personnel d'exploitation spécialisé.

1. Ce document a été rédigé en utilisant les règles de la réforme du français 1990 dont les règles suggèrent, entre autres, la soudure d'un certain nombre de mots composés. C'est pour cela que nous aurons, par exemple, *plateforme* au lieu de *plate-forme*. Toujours conformément à ces règles, les « i » et les « u » s'écriront sans accent circonflexe notamment dans des mots tels que « connaitre » (au lieu de « connaître ») et « cout » (au lieu de « coût »).

Dans le cadre de cette thèse, nous allons étudier un modèle où la plateforme de calcul sous-jacente notamment de type grappe de calcul est mutualisée entre plusieurs organisations. Pour ce modèle qui vise en particulier des entreprises de petite ou moyenne taille, chacune des organisations doit contribuer pour l'acquisition, le fonctionnement et l'exploitation de la plateforme. Ce qui a pour vocation à réduire l'impact sur leur investissement respectif.

Dans la suite du chapitre, nous reviendrons d'abord sur le contexte industriel (section 1.1) puis décrirons les objectifs ainsi que les contributions (section 1.2) de la thèse. La section 1.3 présentera l'organisation des chapitres suivants.

1.1 Contexte industriel

Soutenu par le pôle de compétitivité Minalogic, Ciloe [2] est un projet collaboratif visant à accompagner un ensemble de petites et moyennes entreprises [3] éditrices de logiciels de CAO [4] électronique et embarquées dans la mise en œuvre d'un nuage de calcul à la demande. Ces entreprises pourront ainsi fournir à leurs clients des prestations qui s'appuient sur l'utilisation d'applications suivant le modèle SaaS pour améliorer leur compétitivité.

Afin de leur permettre d'avoir accès malgré leur capacité d'investissement faible aux ressources de calcul intensif nécessaires, le projet propose une approche où le nuage repose sur une plateforme de type grappe calcul partagée entre des entreprises. Celles-ci doivent alors chacune investir pour l'acquisition, le fonctionnement et l'exploitation.

Un des objectifs du projet auquel nous nous sommes intéressés durant cette thèse est de développer des méthodologies et outils pour atteindre ce but. C'est-à-dire des outils pour mettre en œuvre un nuage de services de calcul intensif à la demande sur des ressources mutualisées. Cela soulève comme nous le verrons ci-dessous différents problèmes de gestion de ressources.

1.2 Objectifs et contributions

Nous présenterons d'abord les objectifs et ensuite les contributions. Mais avant d'entrer plus en détail dans cette description, nous allons fixer une terminologie que nous emploierons tout au long du document.

Un *service* est défini comme le traitement d'un jeu de données par une application de calcul spécifique.

Une *requête* est une demande de service faite par un client.

Une *tâche* est l'entité chargée de l'exécution d'une requête, et consiste à traiter le jeu de données avec l'application concernée.

2. http ://ciloe.minalogic.net
3. Nous pouvons citer EdXact, Probayes et Infiniscale. La liste complète est disponible sur le site du projet.
4. CAO : Conception Assisté par Ordinateur.

1.2.1 Objectifs

Nos objectifs sont divers et portent sur les problématiques suivantes :

Gestion dynamique et transparente de tâches : Pour exécuter une tâche, la préparation et le déploiement de l'environnement d'exécution qui inclut l'allocation de ressources ainsi que l'exécution proprement dite de la tâche, doivent être automatiques et aussi transparents que possible pour le client. Il en est de même de la libération des ressources allouées pour cette exécution qui doit être également automatique et transparente.

Exécution simultanée de tâches et cloisonnement : Pour mieux exploiter la capacité disponible, il parait indispensable de pouvoir exécuter plusieurs tâches simultanément sur un nœud. En effet, il est fréquent que le nombre de processeurs (ou cœurs) disponibles sur un nœud de calcul soit supérieur aux capacités requises par une tâche. Toutefois, une telle exécution doit garantir certaines propriétés. Notamment les tâches s'exécutant sur un nœud donné doivent être cloisonnées vis-à-vis des autres. Cela signifie que chacune d'elles doit avoir accès de manière indépendante et exclusive aux ressources et aux données qui ont été affectées pour son exécution.

Partage proportionnel et utilisation efficace de ressources : Nous devons répondre à un compromis entre l'équité et l'utilisation. En effet, chaque entreprise investit sur la plateforme et doit s'attendre à ce que ses services puissent utiliser un ratio de ressources proportionnel à cet investissement. Mais au delà de ce besoin de partage, il est également nécessaire de maximiser l'utilisation des ressources en évitant par exemple de laisser des tâches en attente alors qu'il y a des ressources inutilisées. Cette gestion de ressources doit par ailleurs limiter l'impact d'un comportement égoïste qui ne motiverait pas la mutualisation. Pour illustration, considérons une situation où par rapport à leurs besoins respectifs certaines entreprises ont sous-investi tandis que d'autres ont investi plus ou moins convenablement à leur besoin. Un comportement égoïste peut s'expliquer par le fait que des entreprises qui ont sous-investi surexploitent les ressources au détriment des autres.

1.2.2 Contributions

Nos contributions répondent aux différents objectifs et s'organisent autour des points suivants :

Architecture virtualisée pour l'exécution dynamique de tâches : Nous avons défini une architecture où les tâches sont exécutées au sein de machines virtuelles. Ce choix de la virtualisation est essentiel car, d'une part, les machines virtuelles fournissent un support pertinent pour partitionner un nœud entre plusieurs tâches tout en garantissant un niveau de cloisonnement (ou d'isolation) élevé entre des tâches s'exécutant au sein de machines virtuelles distinctes. D'autre part cette virtualisation sert de socle pour la gestion dynamique de tâches, les machines virtuelles apportant une souplesse et plusieurs autres propriétés qui permettent de créer et configurer automatiquement les environnements d'exécution des tâches.

Notre architecture intègre une composante spécifique appelé gestionnaire SaaS qui assure de manière dynamique et transparente, l'allocation de ressources, la création, la personnalisation, le déploiement des environnements d'exécution (machines virtuelles) ainsi que le démarrage effectif des tâches au sein de ces environnements. Pour gérer le cycle des machines virtuelles sur la grappe, cette composante qui joue également le rôle de gestionnaire de ressources, s'appuie sur un gestionnaire d'infrastructure virtuelle (OpenNebula) pour nous éviter d'avoir à redévelopper des fonctionnalités de virtualisation déjà testées et éprouvées dans un tel outil.

Partage flexible de ressources : Nous avons proposé une gestion de ressources flexible qui s'appuie sur des propriétés de la virtualisation. La politique d'affectation de ressources repose sur une notion de bail de ressources associée aux machines virtuelles dans lesquelles les tâches s'exécutent. La durée d'un bail est variable, elle débute au démarrage de la tâche à laquelle il est affecté et s'achève lorsque cette dernière se termine. Brièvement, le mécanisme consiste pour le gestionnaire SaaS à assigner spécifiquement des baux aux tâches de manière à satisfaire convenablement les objectifs de partage et d'utilisation ressources.

Prototypage et validation expérimentale Cette thèse se déroulant dans un contexte collaboratif avec des partenaires industriels, nous avons validé nos différentes contributions à partir d'une mise en œuvre. Pour cela, nous avons d'abord développé un prototype du gestionnaire SaaS (SVMSched) en nous appuyant sur OpenNebula qui est un outil ouvert et performant pour manipuler des machines virtuelles sur des infrastructures distribuées. A partir de ce prototype, nous avons mené des expériences dont les résultats ont permis de valider les contributions.

En outre, nous avons à partir d'un ensemble d'applications *Benchmark* étudié sur notre architecture les performances de divers types d'applications selon leurs exigences en ressources CPU, en ressources mémoire ou en ressources d'entrée-sortie disque.

Ces travaux ont fait l'objet de communications scientifiques :

- Dans [52], nous traitons des mécanismes d'allocation et de partage de ressources dans un nuage avec des ressources partagées entre plusieurs services. Le papier aborde également des questions de performances des applications sous-jacentes ainsi que des problématiques d'optimisation de l'utilisation de ressources.
- Les papiers [54] et [55] portent sur une mise en œuvre. Complémentaires, ces papiers présentent l'architecture logicielle, le mécanisme de partage de ressources et le prototype [5]. Une version étendue de ces travaux a été publiée au sein du livre *Cloud Computing and Services Science* chez Springer-Verlag [53] [6].
- SVMSched le prototype du gestionnaire SaaS que nous avons développé est référencé par le projet OpenNebula comme outil complémentaire [7]. Il a en outre fait l'objet d'une communication sur le blog officiel du projet [51] [8].

5. https ://gforge.inria.fr/projects/svmsched/
6. http ://www.springer.com/computer/swe/book/978-1-4614-2325-6
7. http ://opennebula.org/software :ecosystem/
8. http ://blog.opennebula.org/ ?p=1646

– Enfin, nos développements durant cette thèse nous ont valu le premier prix du Challenge Grid'5000 organisé en Avril 2011 à Reims [9].

1.3 Organisation du document

Le chapitre 2 dresse un état de l'art sur le calcul dans le nuage et le calcul intensif. Nous analyserons notamment les évolutions des technologies de calcul en montrant comment on est passé du calcul sur des grappes au calcul dans le nuage. Dans le chapitre 3, nous étudierons la question de la mutualisation de ressources avec les problématiques de gestion de ressources que cela soulève. Dans le chapitre 4, nous introduirons les idées fondamentales de la solution qui a été proposée pour répondre à ces problèmes. Les chapitres 5 et 6 seront consacrés à la mise en œuvre. Le chapitre 5 présentera l'architecture proposée, tandis que le chapitre 6 décrira le prototype du gestionnaire SaaS. Le chapitre 7 décrira les expériences, ainsi que les résultats qui ont permis de valider nos contributions. Le document se terminera par une conclusion et des idées pour des travaux futurs.

9. https ://www.grid5000.fr/mediawiki/index.php/Grid5000 :News

Calcul à grande échelle : des systèmes parallèles aux nuages de calcul 2

Le calcul à grande échelle couvre un grand champ d'usages tels que le calcul haute performance, les systèmes repartis et le calcul dans le nuage. De manière générale, nous caractérisons un environnement de calcul suivant trois aspects fondamentaux illustrés sur la figure 2.1 :

Applications : Une application désigne un programme ou un logiciel permettant de résoudre un problème donné au moyen de traitements informatiques dont les besoins en puissance de calcul peuvent être plus ou moins importants.

Outils d'exploitation : Concernant par exemple l'ordonnancement de tâches et la gestion de ressources, ils désignent l'ensemble des outils logiciels employés pour allouer les ressources aux applications avec pour objectif d'utiliser efficacement la puissance de calcul délivrée par les ressources matérielles.

Ressources matérielles : Elles incluent entre autres les serveurs ou nœuds de calcul, les dispositifs de stockage et les équipements d'interconnexion réseaux qui fournissent les capacités nécessaires au traitement des applications.

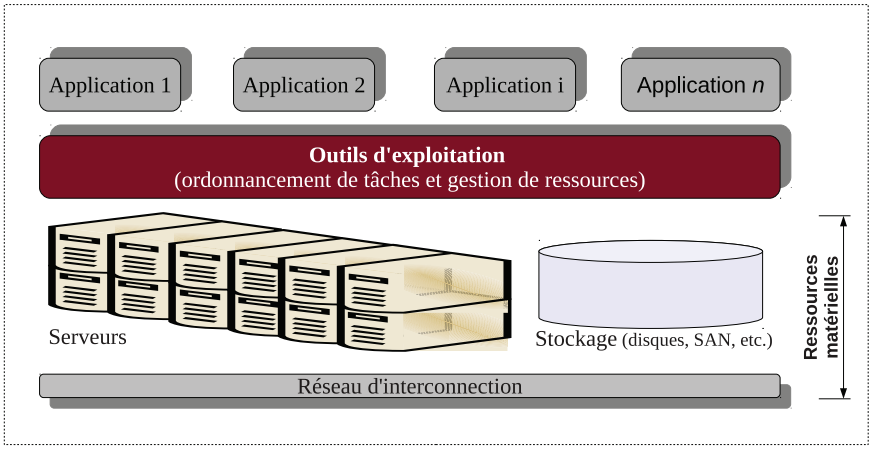

FIGURE 2.1 – Vue générale d'une plateforme de calcul.

Dans ce chapitre, nous étudierons le calcul à grande échelle sous ces différents aspects. Nous insisterons davantage sur les aspects exploitation qui nous intéressent particulière-

ment dans cette thèse. La prochaine section fera un tour d'horizon en ce qui concerne le calcul haute performance et le calcul dans le nuage. La section 2.2 étudiera les principales architectures matérielles des plateformes de calcul. Avant de conclure le chapitre, la section 2.3 étudiera les problèmes usuels de gestion de ressources, ainsi que les mécanismes développés pour les traiter. Dans la section 3.4 enfin, nous ferons une synthèse pour mieux situer nos objectifs.

2.1 Contextes des plateformes de calcul

Depuis la création des premiers ordinateurs, les technologies de calcul à grande échelle ont fortement évolué avec les besoins des applications. Motivée par des besoins de calcul haute performance et/ou de calcul à plus ou moins grande échelle, cette évolution a été marquée par l'apparition de nouvelles architectures de traitement. Des ordinateurs monoprocesseur à l'origine, nous sommes passés à des architectures parallèles et distribuées telles que les grilles avec des capacités de traitement accrues.

Dans cette section, nous étudierons les besoins des plateformes calcul sous deux angles : d'abord sous un angle calcul intensif et ensuite d'un point de vue de calcul dans le nuage.

2.1.1 Calcul haute performance

Le calcul haute performance [69] désigne l'ensemble des méthodes et mécanismes tant au niveau applicatif que matériel utilisés pour réduire le temps d'exécution d'une application. Historiquement, il a été poussé par un besoin toujours plus accru en puissance de calcul des applications en calcul scientifique.

Le calcul scientifique [71], utilisé en ingénierie et en science, désigne l'ensemble des méthodes permettant d'étudier numériquement des systèmes et des phénomènes physiques complexes. C'est souvent le cas de systèmes ou de phénomènes nouveaux ou existants dont la représentation ou la reproduction est difficile, voire impossible. L'étude de ces problèmes passe par des simulations et des optimisations sur des modèles numériques.

Ces différentes étapes nécessitent beaucoup de temps de traitement. Pour illustrer, prenons un exemple concernant la construction d'un nouveau circuit intégré. A l'ère de la miniaturisation, on doit créer des circuits intégrés de plus en plus petits mais qui doivent être toujours plus performants et robustes pour fonctionner dans des conditions extrêmes (ex. chocs, températures basses ou élevées). Comme illustré sur la figure 2.2, on doit concevoir un modèle du circuit et vérifier son comportement en cours de fonctionnement par simulation dans ces différentes conditions. Ce processus est répété tant que le modèle n'est pas validé pour être fabriqué en série.

Les différentes étapes de traitement sont consommatrices en temps et en espace mémoire. Elles nécessitent des ressources matérielles appropriées, ainsi que des optimisations au niveau applicatif.

Pour être plus concret, considérons par exemple l'application Jivaro [1] qui intervient dans

1. Jivarod (http ://www.edxact.com/products/jivaro) est une application de EdXact une des PMEs partenaires du projet Ciloe.

une telle chaine de simulation. L'enjeu des vérifications étant de réduire le temps avant la mise sur le marché, et donc le cout de fabrication du circuit. Par exemple l'application Jivaro a pour but de réduire le temps de ces vérifications. Même si elle atteint globalement cet objectif, elle a besoin de capacités de traitement et de mémoires importantes pouvant aller jusqu'à plus de 10 heures de calcul et jusqu'à 50 gigaoctets de mémoire vive. Avec les nouvelles générations de circuits, l'éditeur prédit même des futurs besoin en mémoire vive autour des 100 gigaoctets.

Des exemples comme celui-ci sont légions dans les différents domaines de l'ingénierie et de la science. En épidémiologie, météorologie ou même en aéronautique pour ne citer que ces domaines, les besoins en capacité de calcul sont tout aussi marqués.

Dans ce contexte, le calcul haute performance apporte des solutions tant au niveau des applications que des plateformes matérielles de calcul. Ces solutions, abordées dans la section 2.3, couvrent entre autres l'optimisation, la parallélisation des applications (section 2.3.1) ainsi que la définition de nouvelles architectures de calcul performantes. Dans la section suivante, nous allons étudier les enjeux du calcul à grande échelle dans le domaine de calcul dans le nuage.

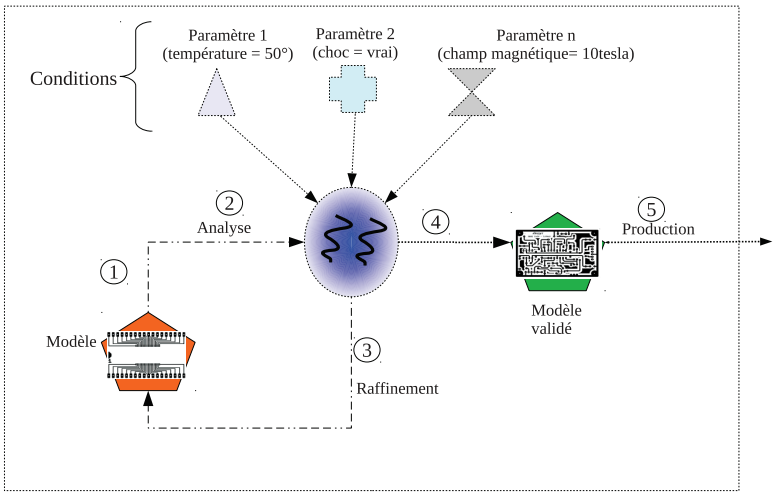

FIGURE 2.2 – Exemple d'un problème nécessitant des calculs : simulation d'un circuit électronique.

2.1.2 Calcul dans le nuage

Dans la littérature [126, 121, 34, 36], on retrouve plusieurs définition du calcul dans le nuage. En plus du fait que le concept soit assez récent, cette multitude de définitions est due au fait que le concept a émergé de l'industrie avec une grande diversité d'offres et de slogans commerciaux.

Pour mieux le comprendre, nous proposerons une définition assez généraliste en présentant ses concepts fondamentaux pour enfin comprendre ses enjeux concernant le calcul à grande échelle.

2.1.2.1 Concepts fondamentaux de nuages de calcul

Plus connu sous le terme *cloud computing* [126, 121, 34, 36], le calcul dans le nuage désigne un modèle de traitement où les ressources informatiques comme les capacités de calcul, de stockage de données ou même les logiciels, sont déployées à distance par un fournisseur et rendues accessibles comme des services à la demande.

Typiquement payants mais parfois gratuits, ces services ont pour but de remplacer les capacités matérielles, les logiciels ou les données traditionnellement acquis(es) et maintenu(e)s par des organisations leur propre frais. Lorsque l'utilisation du service est payante, la plateforme doit offrir un mode de facturation souple qui permette à l'utilisateur de ne payer que pour ce qu'il a consommé : c'est le paiement à l'utilisation ou *pay-as-you-go*.

La figure 2.3 illustre une infrastructure de nuage de calcul. Le client qui accède au service via un réseau ou Internet n'a aucune vision de la manière dont le service est mis en œuvre (vision de nuage). Il « ne gère » aucune considération liée au développement, au déploiement, ou à l'exploitation des ressources matérielles et logicielles qui sous-tendent le service. Toutes ces considérations sont à la charge du fournisseur du service. Contrairement au modèle traditionnel des systèmes d'information où l'acquisition du matériel et des logiciels, leur installation et leur exploitation sont assurées par chaque organisation.

Originellement, le calcul dans le nuage vise à répondre à deux problèmes d'antan dans le domaine du calcul à grande échelle : (i) permettre le traitement à (très) grande échelle, (ii) en simplifiant l'accès et l'utilisation de ressources de calcul distantes et distribuées, de manière aussi transparente que possible. C'est pour cette raison que le calcul dans le nuage est considéré par certains [44, 48] comme un pendant commercial du calcul sur grilles [66, 38].

Mais en pratique, cette vision du nuage où c'est le fournisseur qui s'occupe de toutes les tâches d'administration est plutôt partielle. Pour illustration, pour garantir qu'une défaillance d'un serveur virtuel n'aura pas de conséquences désastreuses sur ses applications, chaque client doit à son niveau mettre en œuvre un mécanisme de résilience approprié via des sauvegardes régulières par exemple

Besoin en puissance de calcul

Pour satisfaire la demande, une infrastructure de nuage nécessite une capacité de traitement adéquate.

En effet, un service de nuage doit satisfaire deux propriétés essentielles : l'accès au service doit être *à la demande* et *élastique*. Autrement dit, le service est rendu disponible à la demande du client et, pendant que ce dernier utilise le service, il peut spontanément demander plus ou moins de ressources et la plateforme doit pouvoir répondre à ces exigences. Un nuage de calcul doit également répondre aux éventuels pics de demande en donnant notamment l'illusion aux utilisateurs de disposer de ressources infinies.

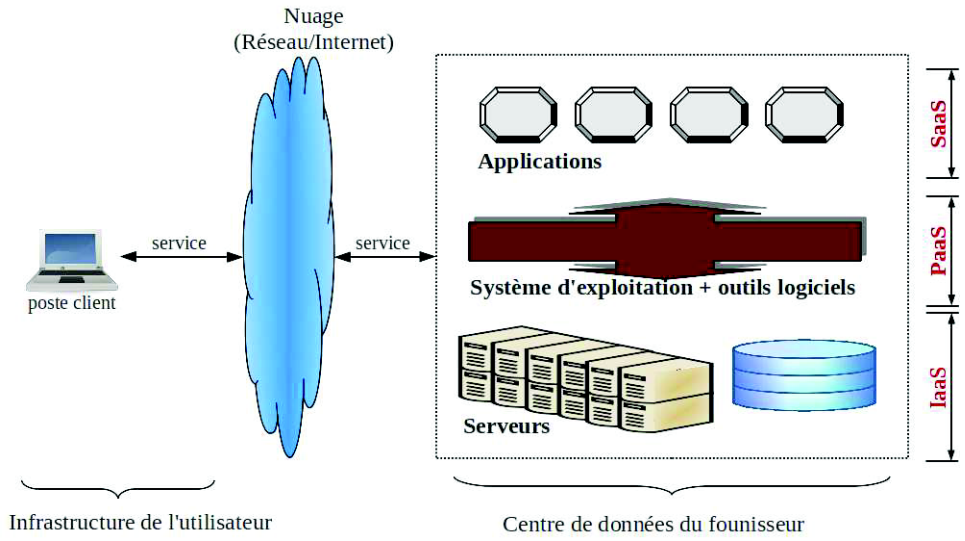

FIGURE 2.3 – Aperçu d'une plateforme de calcul dans le nuage.

2.1.2.2 Modèles de services

Comme indiqué, les services de nuage reposent sur divers types de ressources incluant le matériel, les logiciels et/ou les données. Le potentiel de services que l'on peut proposer est ainsi important. Pour les distinguer, il existe plusieurs taxinomies ou nomenclatures dont l'essentiel sera présenté dans cette section.

Nomenclature NIST

L'institut de standardisation (*National Institue of Standard and Technology*) distingue trois modèles de service [36], selon que le service est basé sur le logiciel, l'infrastructure matérielle ou une combinaison des deux :

Service basé sur des ressources matérielles ou *Infrastructure as a Service* (IaaS) : le service consiste en l'accès et l'utilisation de capacités matérielles destinées par exemple au calcul, au stockage ou au transfert de données en réseau.

Les offres du marché sont importantes et peuvent difficilement être listées exhaustivement, d'autant que de nouvelles offres ne cessent de paraitre. Amazon [1], GoGrid [12] ou encore Rackspace [25] font partie des acteurs majeurs avec leurs plateformes pionnières. Dans ces plateformes, la facturation des capacités de calcul se fait à l'heure, le prix unitaire dépend de la puissance du serveur virtuel alloué. La facturation des capacités de stockage, quant à elle, est fonction de l'espace de stockage utilisé – l'unité de facturation étant le gigaoctet.

Service basé sur des ressources matérielles et logicielles ou *Platform as a Service* (PaaS). Le service consiste à fournir à la fois des capacités matérielles (serveurs, stockage...) et un environnement logiciel intégré pour le développement, le déploiement ou l'hébergement d'applications. Une telle plateforme réduit considérablement la complexité et le temps liés aux tâches d'administration pour les développeurs.

Par exemple, Google App Engine [13], Force.com [10] et Microsoft Azure [19], trois offres PaaS majeures du marché, fournissent des environnements matériels (serveurs, stockage...) et logiciels intégrés pour le développement, le déploiement, l'hébergement et l'exploitation d'applications Web. La suite de logiciels intégrés fournie dans un environnement peut couvrir toutes les couches d'une application Web : couche présentation, couche métier et couche logique.

Service basé sur des ressources logicielles ou *Software as a Service* (SaaS). Le service consiste en l'accès et l'utilisation d'un logiciel [120, 100, 77]. Le logiciel est déjà déployé et prêt à l'utilisation. Le service est en général fourni par l'éditeur du logiciel et l'accès se fait via un outil client, typiquement un navigateur Web. C'est par exemple le cas des offres Google Apps [14], Office365 [21] ou Saleforce.com [26] – trois des plus populaires du marché.

Autres nomenclatures

La taxinomie NIST est une déclinaison immédiate de la vision des systèmes d'information traditionnels qui se focalise sur les aspects logiciel, plateforme et infrastructure d'un environnement de calcul. C'est la raison pour laquelle elle est appelée modèle SPI (*Software, Platform, Infrastructure*).

Elle permet de classer aisément les premières plateformes de nuage citées précédemment. Cependant, nous voyons apparaitre de nouvelles plateformes où les services sont basés sur de nouveaux types de ressources qui ne peuvent être intuitivement classées dans un modèle SPI. Ces offres concernent entre autres l'accès à des données d'archive (magazines, journaux, statistiques de vente en ligne...), l'utilisation de solution de VoIP, etc. La taxonomie NIST parait peu adaptée et limitée pour ces nouveaux services.

Pour l'étendre, plusieurs autres taxinomies ont été proposées [130, 78, 88, 68] (*Database-as-a-Service, Communication-as-a-Service, Data-as-a-Service...*). Les présenter exhaustivement sort du cadre de ce document. Les lecteurs intéressés pourront consulter les documents référencés ci-dessus. Aussi, la page d'Amazon Web Service [1] nous donne un aperçu de la diversité des modèles de service.

En particulier, nous décrirons le modèle *Hardware-as-a-Service* (HaaS), qui peut-être vu comme un raffinement du modèle IaaS du NIST. Il se situe en effet à un niveau plus bas que le IaaS et consiste à fournir des capacités brutes d'une ressource informatique. Comparé à un IaaS tel que Amazon EC2, la vision HaaS se démarque par le fait que, au lieu des machines virtuelles, ce sont des nœuds réels ou physiques qui sont alloués. Grid'5000 [15], une grille dédiée à l'expérimentation en France, est un bon exemple de cette vision. Dans Grid'5000 en effet, lorsque l'utilisation réserve un nœud, c'est un serveur brut (pas une machine virtuelle) qui lui est alloué. Il a alors la possibilité de déployer le système et ensuite les logiciels qu'il souhaite pour réaliser ses expérimentations.

Plus généralement, la littérature [82, 29] définit un modèle abstrait, *Anything as a Service* noté *XaaS*, comme une extrapolation de cette diversité de services dans un nuage où diverses ressources peuvent constituer le socle d'un service. Ce modèle généraliste permet de classer rapidement et aisément tout nouveau service qui répond aux caractéristiques de base d'un

service de nuage (accès distant, à la demande et élasticité).

A regarder de près, ces nouvelles taxinomies ne sont pas en opposition avec la nomenclature NIST. En effet, dans sa définition, NIST définit un nuage de calcul en cinq points essentiels [36], qui sont vérifiables par ces nouveaux modèles de nuage :

- *Accès au service à la demande :* un client peut selon ses besoins demander et doit obtenir plus de capacités, et cela doit se faire automatiquement sans interaction humaine venant du fournisseur de service.
- *Accès au service via le réseau :* le service est accessible via des mécanismes standards de manière à favoriser l'utilisation d'outils réseau hétérogènes. Ainsi, l'application cliente peut être de type client lourd ou client léger tout comme le terminal d'accès peut être un ordinateur ou un téléphone.
- *Élasticité réactive/rapide :* le service doit être élastique et rapide, c'est-à-dire que l'allocation et la désallocation de ressources doivent être automatiques de manière à s'ajuster rapidement à une forte ou faible demande.
- *Service mesurable :* les systèmes gérant le nuage doivent être capable de contrôler et mesurer l'utilisation de ressource en collectant des métriques à des niveaux d'abstractions appropriés au type de service offert. Les métriques peuvent par exemple concerner l'espace de stockage utilisé, la charge des processeurs, la bande passante disponible ou utilisée, le nombre d'utilisateurs actifs, etc.
- *Mutualisation de ressources :* les ressources de traitement sous-jacentes sont typiquement mutualisées pour servir plusieurs clients, toujours en permettant une allocation et une desallocation automatiquement pour s'adapter rapidement aux demandes des clients.

2.1.2.3 Types de nuages

La littérature [36] distingue quatre types de nuages selon le public qui peut avoir accès au service : un service de nuage peut être accessible au grand public ou seulement à un ensemble restreint d'entités (organisations ou personnes) :

Nuage public ou *public cloud* : le service est accessible via Internet à toute entité (personne ou organisation) qui le souhaite. C'est le cas des plateformes comme DropBox [8] et toutes les autres que nous avons citées jusqu'à présent.

Nuage privé (*private cloud, on-premise cloud* ou encore *corporate cloud*), où le service est accessible seulement à un groupe restreint de personnes, notamment au sein d'une organisation ou entreprise. Un exemple est donné avec le nuage d'Intel [75], qui fournit des capacités de calcul à la demande pour les ingénieurs de l'entreprise.

Nuage communautaire ou *community cloud* : c'est un nuage mis en place et exploité par une corporation d'organisations pour répondre à des besoins communs. Par exemple, le nuage Apps.Gov [2] a été mis en place par l'administration américaine pour fournir à la demande des capacités de calcul et d'hébergement pour ses entités.

Nuage hybride (*hybrid cloud*), qui désigne un nuage qui combine les technologies et/ou les services provenant de différents types de nuages. On retrouve généralement ce type de nuage dans des entreprises et des organisations, qui couplent les ressources de leur

nuage privé avec d'autres provenant de nuages publics ou communautaires afin de répondre à de fortes charges (demandes) ne pouvant être soutenues par les ressources à leur disposition.

2.2 Organisations matérielles d'une plateforme de calcul

Notre étude s'organisera autour de deux types de systèmes :

Systèmes à mémoire centralisée : Ils englobent des ordinateurs ou serveurs de calcul regroupant, chacun, plusieurs processeurs qui se partagent la même mémoire principale. Le nombre de processeurs et la taille de la mémoire peuvent être très importants. Pour illustration, les serveurs de la gamme Altix UV de SGI ont jusqu'à 2560 cœurs et 16To de mémoire.

Systèmes distribués : Ce sont des systèmes composés d'un ensemble d'ordinateurs repartis sur une petite ou une large échelle géographique et interconnectés via un réseau (LAN ou Internet), de manière à coopérer pour exécuter des applications. Les processeurs d'un même nœud se partagent la mémoire principale de ce dernier mais, de part leur distribution, des processeurs situés sur de nœuds distincts ne communiquent que par échanges de messages réseaux.

Dans la suite, nous nous intéresserons davantage aux systèmes distribués qui se rapprochent plus des problématiques étudiées dans cette thèse.

2.2.1 Grappes d'ordinateurs

Une grappe est un groupe d'ordinateurs fortement liés via un réseau local [99] qui fonctionne ensemble comme un superordinateur parallèle (figure 2.4). Appelé nœuds, les ordinateurs constituant une grappe sont en général homogènes aussi bien d'un point de vue matériel que de l'environnement logiciel. Mais dans certains cas le matériel grappe peut être hétérogène.

Le concept de grappe est né du projet Beowulf [117], une des premières grappes de calcul qui était constituée de 16 ordinateurs monoprocesseurs de 100 Mhz de fréquence interconnectés sur un réseau de 10 mégabits Ethernet.

Les nœuds des grappes contemporaines sont de plus sophistiqués et puissants, typiquement des ordinateurs multiprocesseurs à mémoire partagée (SMP, NUMA, GPU, etc.) interconnectés via des réseaux rapides de type Gigabit Ethernet (GbE) ou Infiniband.

Ainsi présentée, nous avons considéré une grappe où les nœuds sont en permanence dédiés au calcul. Mais avec des parcs informatiques qui ont été de plus en plus enrichis avec des ordinateurs sophistiqués au niveau des stations de travail, une nouvelle catégorie de grappe a émergé. C'est en particulier le cas des grappes de ressources conçues à partir des stations de travail d'un intranet [129]. L'objectif d'une telle grappe est d'exploiter les heures d'inactivité de ces stations (pendant les pauses médianes, la nuit, les weekends...) pour exécuter des tâches de calcul. Nécessitant des outils spécialisés tels que Condor [89] ou encore ComputeMode [2], le basculement des stations de travail entre le mode grappe et le mode

2. http ://sari.grenoble-inp.fr/spip.php ?article80

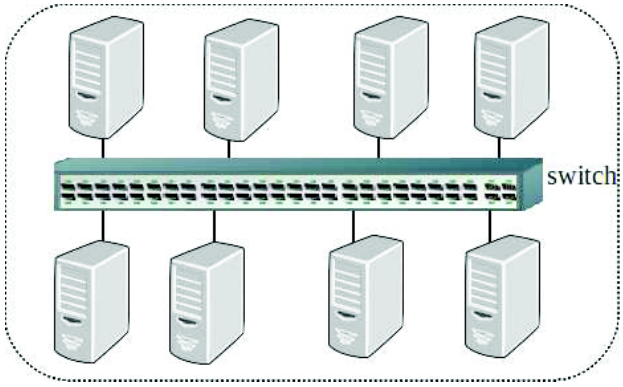

FIGURE 2.4 – Grappe d'ordinateurs

station de travail peut être programmée sur des dates (jour/heure) ou automatiquement cadrée sur les périodes de veille des différentes stations.

Dans certains cas, les nœuds d'une grappe peuvent être des machines virtuelles [115]. Ce sont des grappes de machines virtuelles aussi appelées grappes virtuelles où les machines virtuelles sont interconnectées via un réseau virtuel qui fonctionnent au dessus d'un réseau d'interconnexion réel. Voir une illustration sur la figure 2.5.

Une grappe virtuelle vise à faciliter l'utilisation de ressources d'une infrastructure distribuée en masquant l'hétérogénéité et la distribution de ressources grâce à la virtualisation (matériel, système d'exploitation et logiciel). Le projet RESERVOIR (*Resources and Services Virtualization Without Barriers* [108]), par exemple, utilise le concept de grappe virtuelle pour mettre en œuvre et gérer des services informatiques complexes à partir de ressources physiques hétérogènes et distribuées.

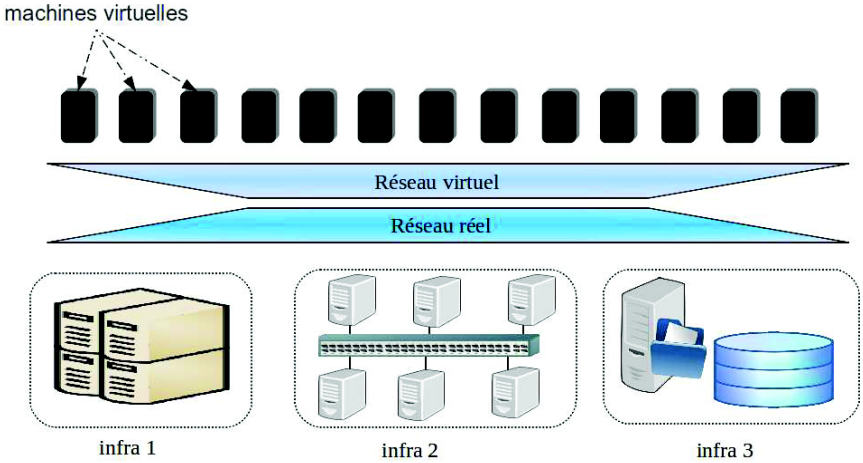

FIGURE 2.5 – Grappe de machines virtuelles distribuées sur trois sites géographiques.

2.2.2 Grilles

Une grille [66, 38] est un ensemble de ressources de traitement (grappes, ordinateurs NUMA, périphériques de stockage, etc.) hétérogènes et distribuées interconnectées au

moyen d'un réseau métropolitain ou Internet pour former une infrastructure de traitement de très grande taille.

Les ressources sont regroupées en *site* selon leur localité géographique (figure 2.6). Par exemple, Grid'5000 est composée de 12 sites (10 sites en France, 1 site au Luxembourg et 1 site au Brésil). Les différents sites d'une grille peuvent appartenir à des organisations distinctes et peuvent également être hétérogènes, aussi bien au niveau matériel que logiciel. Malgré cette distribution, une grille vise à offrir une vue cohérente de l'ensemble de ses ressources et donc de faciliter leur utilisation. Même si en pratique cet objectif n'est pas toujours atteint [45].

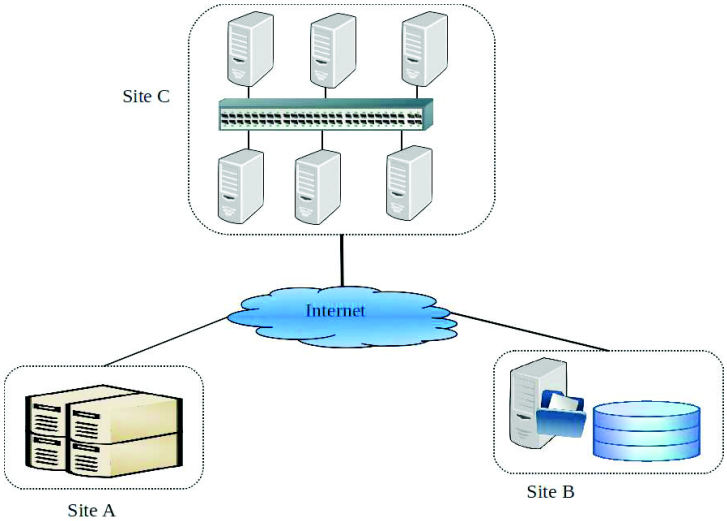

FIGURE 2.6 – Illustration d'une grille constituée de trois sites.

Comme dans le cas des grappes, il existe un cas particulier de grilles où des nœuds ne sont pas en permanence dédiés au calcul et peuvent être inaccessibles d'un moment à l'autre. Comme une grappe volatile, une grille volatile se compose de stations de travail non dédiées reparties sur Internet. Une telle grille est communément appelée *Desktop Grid*.

Reposant sur des outils tels que BOINC [32], une telle grille est généralement utilisée pour faire du calcul bénévole (*Volunteer Computing*) où les capacités de calcul sont fournies par des cycles inutilisés de diverses ressources de calcul (stations de travail, serveurs...) connectées sur un réseau métropolitain ou Internet. SETI@HOME [33], dont le but est d'exploiter la puissance inutilisée des stations de travail à travers le monde pour la recherche d'une intelligence extra-terrestre, est l'un des projets historiques qui ont vulgarisés ce type d'infrastructure.

2.2.3 Systèmes pair-à-pair

Dans tous les systèmes présentés ci-dessus, les ressources sont gérées de manière centralisée par un gestionnaire de ressources. L'inconvénient est que si le gestionnaire de ressources tombe en panne, par exemple, si le serveur qui l'héberge a une défaillance qui le rend indisponible, il n'est plus possible d'utiliser les ressources de la plateforme.

L'objectif d'un système pair-à-pair est d'éviter ce genre problème en proposant un mécanisme qui permet de décentraliser le contrôle d'accès aux ressources. Comme illustré sur la figure 2.7, cela vise à permettre à chaque pair (nœud) de collaborer avec n'importe quel autre pair, sans nécessiterss un gestionnaire de ressources centralisé.

Ces systèmes qui sont plutôt adaptés à des plateformes de partage de fichiers, restent complexe à mettre en œuvre. Même Napster [67], une des plateformes qui a contribué à les populariser, était basée sur un serveur centralisé qui répertorie les pairs. Si ce serveur tombe en panne, le système entier devient indisponible, comme dans le cas des autres plateformes présentées précédemment. Sur cette base, certains considèrent des plateformes de calcul volontaire comme SETI@home (basé sur BOINC) comme des systèmes pair-à-pair.

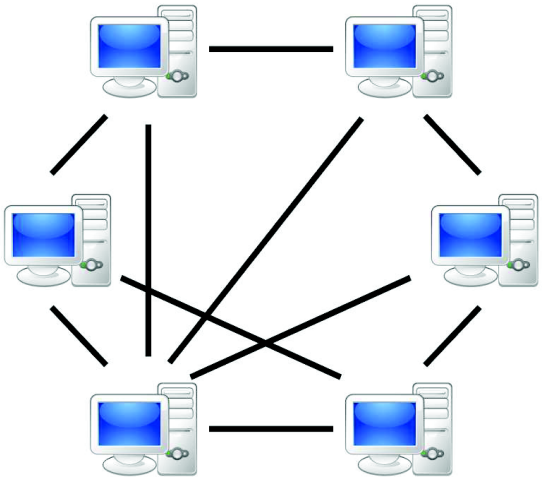

FIGURE 2.7 – Illustration d'un système de pair-à-pair.

2.3 Exploitation d'une plateforme de calcul

Nous considérerons trois aspects d'exploitation d'une plateforme de calcul : l'optimisation et la parallélisation des applications, l'ordonnancement de tâches et la gestion de ressources, et la virtualisation de ressources. Nous insisterons sur ces deux derniers aspects qui nous intéressent particulièrement dans le cadre de cette thèse.

2.3.1 Optimisation et parallélisation des applications

L'optimisation et la parallélisation visent à accoitre les performances d'une application. De plus, elles favorisent une meilleure utilisation des ressources de calcul. En général, lors du développement initial d'une application l'objectif est avant tout tourné vers l'exactitude (*correctness*) ou l'aptitude du code à renvoyer les résultats attendus.

L'optimisation et la parallélisation concernent tout ou partie du code d'une application.

Simplement expliquée, l'optimisation [70] consiste à réorganiser ou à ré-écrire le code en choisissant des algorithmes et des structures de données qui permettent de mieux exploiter

l'architecture matérielle afin de calculer plus vite. L'optimisation ne nécessite pas d'outils spécifiques mais réside plutôt dans la démarche de conception, en général.

Par exemple, nous avons montré dans [52] comment un choix approprié de structure de données contribuerait à réduire de jusqu'à 70% la durée d'exécution et de près de 40% l'occupation mémoire dans une portion de l'application Jivaro.

Un exemple classique d'optimisation des algorithmes concerne les applications d'algèbre linéaire où, une simple réorganisation des boucles ou des tailles des blocs de matrices ou vecteurs – pour mieux exploiter une architecture matérielle donnée (organisation mémoire et taille des caches) – permet d'améliorer de manière très significative les performances de l'application [109].

La parallélisation [84] consiste à décomposer un traitement appelé tâche en sous-tâches réalisables simultanément. Elle est plus complexe et nécessite généralement une coordination entre ces sous-tâches.

La parallélisation sur des architectures parallèles [103] s'appuie en général sur des outils spécifiques de compilation et de programmation. Sur les architectures superscalaire et SMT, par exemple, le parallélisme est détecté et généré par le compilateur (les outils sont mises en œuvre au niveau du compilateur). En revanche, sur une architecture multiprogrammée de types vectoriels, SMP, NUMA ou distribués, le parallélisme est explicitement exprimé par le développeur via des mécanismes spécifiques lui permettant de créer et manipuler des tâches. C'est le cas des bibliothèques basées sur Posix Threads [43], OpenMP [56], CUDA [112], MPI [65], PVM [118] ou encore MapReduce [61, 104].

Dans le domaine de calcul haute performance, les problématiques d'optimisation et de parallélisation des applications sont à juste titre omniprésentes. Elles seraient encore plus importantes dans un contexte de nuage car les services doivent avoir des performances raisonnables pour les clients. Sachant qu'à cause des surcouts de performance induits par la virtualisation, les besoins d'optimisation seraient plus criants : ils vont au delà des applications et concernent également les gestionnaires de machines virtuelles.

2.3.2 Ordonnancement de tâches et gestion de ressources

L'ordonnancement de tâches vise à favoriser une utilisation efficace de la puissance de calcul délivrée par une infrastructure matérielle de traitement. L'ordonnancement de tâches repose sur deux entités fondamentales : les tâches et les ressources.

Tâche : Désigne tout ou partie du programme d'une application. Une application s'exécute sous forme de tâches ; elle peut être constituée d'une tâche unique ou de plusieurs tâches parallèles.

Ressource : Ici, une ressource désigne toute capacité matérielle utilisée pour exécuter une tâche (ex. nœud de calcul, CPU, mémoire, disque, lien réseau...). Dans la suite, nos références à « ressource » renvoient soit à un cœur (notamment sur les architectures multiprocesseurs à mémoire partagée), soit à un nœud de calcul dans une architecture distribuée.

L'ordonnancement consiste alors à décider quand et où (sur quel nœud) démarrer les différentes tâches de manière à atteindre l'efficacité attendue.

2.3.2.1 Politiques d'ordonnancement

L'efficacité d'un ordonnancement s'évalue suivant différents critères qui prennent en compte deux aspects essentiels [40] : l'utilisation de ressource et la satisfaction des utilisateurs :

- Ressources : la politique d'ordonnancement cherche dans ce cas à maximiser l'utilisation des ressources matérielles (comme les processeurs ou la mémoire). Dans le cas des processeurs, cela vise à optimiser la quantité de temps où les ressources sont occupées à calculer. L'algorithme d'ordonnancement est construit pour optimiser des critères comme le temps global de traitement, le débit ou le nombre d'opérations traitées par unité de temps ou la consommation énergétique.
- Utilisateurs : Dans ce cas, la politique d'ordonnancement vise la satisfaction des utilisateurs. Les algorithmes d'ordonnancement optimisent alors un ou plusieurs des critères tels que les temps de réponse et les temps d'attente.

Au regard de ces deux aspects, nous étudierons dans la suite quelques politiques ou stratégies d'ordonnancement de tâches couramment rencontrées.

Premier arrivé, premier servi

Suivant cette politique encore appelée *First-come, First-served* (FCFS), les tâches sont exécutées dans leur ordre de création. Elles sont en effet stockées dans une file d'attente FIFO (*First-in, First-out* ou premier entré, premier sorti), c'est-à-dire une file où les tâches sont stockées en ordre suivant leur date de création.

Sur la figure 2.8, nous avons présenté un exemple où six tâches sont ordonnancées selon la politique FIFO. Nous avons supposé qu'initialement toutes les tâches sont disponibles dans la file d'attente et numérotées suivant leur ordre respectif de création. Chaque tâche est représentée par un rectangle ; la largeur et la hauteur du rectangle sont respectivement proportionnelles à la durée d'exécution et la quantité de ressources (ex. CPU) nécessaire. La valeur maximale des ressources est représentée par la ligne horizontale en pointillés.

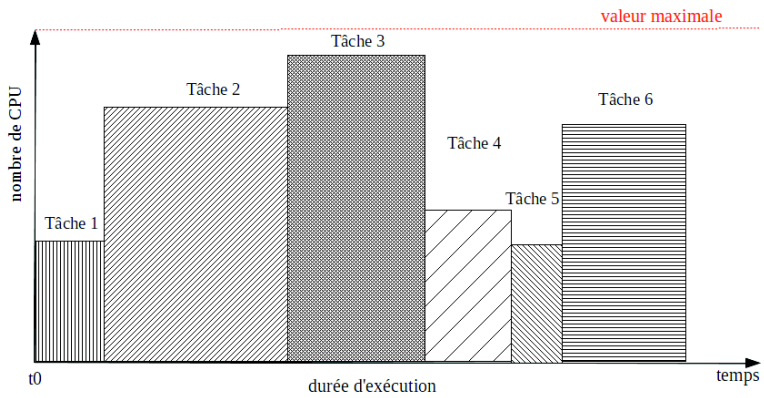

FIGURE 2.8 – Politique premier-arrivé, premier-servi.

Orientée utilisateur et facile à mettre en œuvre, la politique FCFS est inspirée du modèle classique de traitement de file d'attente dans un système traditionnel tel qu'un guichet de banque. Cependant, elle ne permet pas très souvent d'avoir un bon niveau d'utilisation des ressources. Par exemple sur la figure 2.8, on observe que même si les tâches 5 et 6 peuvent être exécutées concurremment avec les tâches 1 et 2, la politique FCFS empêche cela. Leur exécution est alors retardée et les ressources sont alors sous-utilisées.

FCFS avec remplissage (*Backfilling*)

Pour apporter une solution au problème de sous-utilisation de ressources de l'approche classique FCFS, une alternative consiste à autoriser qu'une tâche plus récente demandant une quantité de ressources inférieure ou égale à la quantité de ressources libres puisse être exécutée avant d'autres plus anciennes qui en demandent plus. C'est la politique FCFS avec remplissage *backfilling* [94]. Une illustration est présentée sur la figure 2.9. Le remplissage avec les tâches 4 et 5 permet de mieux utiliser les ressources et de calculer globalement plus vite.

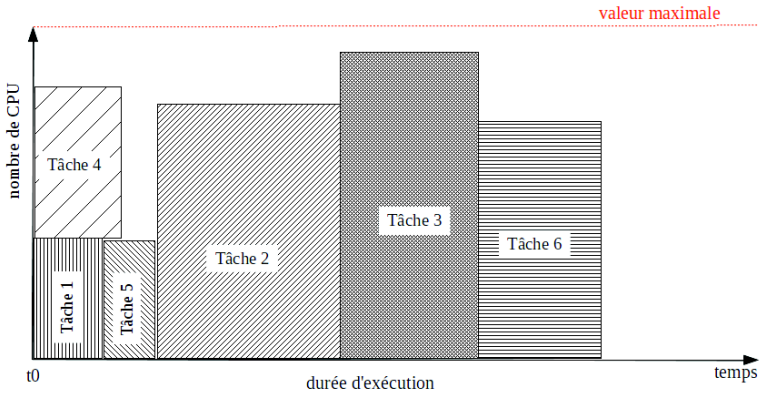

FIGURE 2.9 – Politique premier-arrivé, premier-servi avec remplissage.

Selon la durée d'exécution des tâches, le remplissage peut entrainer un délai supplémentaire concernant le démarrage des plus grosses tâches : cela se produit lorsque la durée d'exécution de la tâche « de remplissage »est plus grande que la période pendant laquelle les ressources devaient rester inutilisées avec une approche FCFS classique. Par exemple sur la figure 2.9, on observe que le remplissage a repoussé la date de démarrage de la tâche 2.

Selon qu'on veuille ou non résoudre ce problème, on distingue deux variantes de remplissage : la première, dite *agressive* et plus connue sous le terme *EASY backfilling*, consiste à appliquer naïvement le remplissage comme sur la figure 2.9. Elle est simple à mettre en œuvre, mais peut retarder le démarrage des grandes tâches comme montré précédemment. Dans un cas non déterministe (où le nombre de tâches n'est pas limité), ce retardement peut être infini [94]. La seconde politique, dite *conservatrice*, évite ce problème en ne faisant passer une tâche que si et seulement si, son exécution ne repoussera pas le démarrage de la tâche qui aurait été démarrée selon une politique FCFS classique.

En conclusion, la stratégie FCFS avec remplissage apporte un compromis entre le délai d'attente des utilisateurs et une meilleure utilisation des ressources.

Ordonnancement avec priorités

Cela consiste à assigner à chaque tâche une priorité, et ainsi, l'ordre d'exécution des tâches est défini par leur priorité. S'il arrivait que plusieurs tâches aient la même priorité, leur ordre d'exécution sera alors déterminé via une politique FCFS.

L'ordonnancement avec priorité vise à privilégier certains utilisateurs ou groupes d'utilisateurs, en réduisant les temps d'attente de leurs tâches. L'ordonnancement guidé par des contraintes économiques [58, 114, 63, 83, 50, 46, 47] est un cas particulier. Il s'agit d'un mécanisme d'ordonnancement où les ressources sont affectées aux tâches des utilisateurs suivant des enchères.

Ce mode d'ordonnancement est assez répandu (on le retrouve dans la plupart des systèmes modernes) mais implique une certaine complexité de mise en œuvre des algorithmes. Notamment parce que les cas d'utilisation sont nombreux et variés. Et par ailleurs si la stratégie de priorité est mise en œuvre naïvement dans un cas non déterministe, l'exécution des tâches de faibles priorités pourrait être retardée significativement, similairement à ce qui se produit avec une politique FCFS avec remplissage agressive. Pour éviter cela, certains ordonnanceurs mettent en œuvre des mécanismes de priorité dynamique où la priorité d'une tâche augmente ou diminue en fonction du temps passé en attente ou à calculer. C'est par exemple le cas dans les systèmes d'exploitation basés sur UNIX ou Linux.

Ordonnancement avec contraintes d'équité

Il peut arriver que, dans une plateforme de calcul où les ressources sont partagées entre les tâches de plusieurs utilisateurs, on ait à garantir aux tâches de certains utilisateurs ou groupes d'utilisateurs des ratios d'utilisation de ressources. C'est l'ordonnancement avec des contraintes d'équité ou (*Fair-share* [80]).

Implémentée dans des systèmes à temps partagé (ex. Unix) et certains gestionnaires de ressources sur des grappes de calcul (ex. maui [41]), cette stratégie repose sur un principe de priorités dynamiques entre les tâches des utilisateurs. Les priorités des tâches de certains utilisateurs ou groupes d'utilisateurs augmentent ou diminuent au cours du temps en fonction de la quantité de ressources utilisées par ces derniers : plus une de ces entités aura utilisé de ressources, moins ses tâches seront prioritaires, et réciproquement.

Pour ne pas sacrifier l'utilisation des ressources, cette politique vise à garantir asymptotiquement sur une fenêtre de temps, dont la largeur est fixée à l'avance, que les tâches d'une entité donnée auront utilisé la quantité ou le quota de ressources auquel elle a droit sous réserve d'une réelle demande.

Ordonnancement préemptif

Toutes les politiques décrites précédemment sont dites non-préemptives. Signifiant que dès qu'une tâche est démarrée elle est exécutée jusqu'à la fin avant que les ressources qui lui ont été allouées soient libérées. Ainsi lorsque toutes les ressources sont allouées, toute nouvelle tâche est mise en attente jusqu'à ce que des ressources soient de nouveau disponibles. C'est-à-dire à la terminaison d'une tâche ou après l'ajout de nouvelles ressources.

Cependant, dans beaucoup de systèmes, certaines tâches ont besoin d'être prises en charge dans un bref délai. Nous pensons notamment à des systèmes interactifs et temps réel. Dans un avion par exemple, la réactivité est essentielle et vitale sinon les conséquences peuvent être catastrophiques (un crash par exemple).

C'est l'objectif visé par un ordonnancement préemptif. Selon une telle politique d'ordonnancement, lorsqu'une tâche ayant une plus forte priorité ou contrainte de réactivité arrive, le système d'ordonnancement doit suspendre ou arrêter l'exécution d'une ou plusieurs tâches ayant de plus faibles priorités pour la faire passer : c'est la préemption. Idéalement, le système devrait suspendre la tâche, sauvegarder son contexte pour la redémarrer plus tard lorsque des ressources seront de nouveau disponibles. Cela nécessite notamment des mécanismes de sauvegarde/reprise et/ou de migration de tâches [92, 111, 101].

Dans un ordonnancement préemptif, une tâche est préemptée si une tâche ayant une plus grande priorité arrive alors qu'il n'y a pas suffisamment de ressources pour l'exécuter.

2.3.2.2 Placement de tâches sur les ressources

Un autre aspect d'ordonnancement de tâches est de déterminer sur quelle ressource (serveur, CPU ou cœur) une tâche sera exécutée. Nous nous intéresserons ici au placement initial des tâches. Les stratégies étudiées ci-après visent divers objectifs (ex. équilibrage de charge) que nous préciserons à chaque fois.

Stratégie gloutonne

Dans une stratégie gloutonne (*greedy strategy* [60]), l'algorithme d'affectation parcourt la liste des ressources et s'arrête dès qu'une ressource pouvant supporter l'exécution de la tâche est trouvée.

C'est une stratégie simple, intuitive et facile à mettre en œuvre. Cependant, elle est peu efficace dans beaucoup de situations, par exemple si l'architecture est hétérogène et qu'une tâche nécessite que l'on tienne compte des spécificités du nœud d'exécution (ex. taille du cache ou de la mémoire vive) pour garantir de performances.

Tourniquai

La stratégie du tourniquai (*round-robin*) place les tâches de manière circulaire sur les ressources. En considérant que les ressources sont stockées dans une liste ordonnancée, si une tâche est placée sur la ressource de rang i, alors la prochaine tâche sera placée sur la ressource de rang $i + 1$.

Cette stratégie vise à favoriser l'équilibrage de charge sur les différentes ressources. Cependant dans un contexte d'économie par exemple, cela n'est pas efficace parce que l'on préférera utiliser le moins de ressources possible.

Appariement

Une stratégie d'appariement (*matchmaking*) [106] est basée sur les affinités entre les tâches et les ressources. Les affinités peuvent s'appliquer sur différents aspects liés par exemple

aux :

- Données, en intégrant notamment la proximité ou la localité des données avec l'application : dans le cas d'applications distribuées, il serait souhaitable de placer une tâche sur le nœud dont le temps de transfert de données est le plus faible possible.
- Ressources matérielles : pour illustration, dans une architecture distribuée et hétérogène, il serait par exemple intéressant d'exécuter une application OpenMP sur un nœud SMP au lieu d'un nœud NUMA. Les affinités peuvent également concerner divers autres facteurs comme la taille de la mémoire vive ou du cache, l'espace disque disponible, le nombre, la fréquence et les taux d'utilisation des processeurs.
- Logiciels : par exemple, dans une architecture distribuée hétérogène comme précédemment, il s'agira de placer certaines tâches sur des nœuds fonctionnant sur un système d'exploitation spécifique.
- Restrictions ou règles d'accès aux ressources : dans une grille par exemple, un site peut avoir des restrictions concernant les tâches d'un ou plusieurs (groupes) utilisateurs.

2.3.3 Virtualisation de ressources

La virtualisation consiste à utiliser une version émulée ou simulée (version virtuelle) d'une ressource (ex. un serveur) au lieu de la version réelle. Dans un centre de données, la virtualisation se symbolise par l'utilisation de machines virtuelles comme environnements d'exécution des applications au lieu d'un système d'exploitation fonctionnant directement sur un vrai matériel [115, 123, 35].

Dans une infrastructure distribuée telle qu'une grappe, la gestion de la virtualisation se situe à deux niveaux : d'abord au niveau des serveurs individuels et ensuite, au niveau distribué. Nous aborderons ces différents aspects ci-dessous.

2.3.3.1 Comprendre la virtualisation

Virtualiser un serveur [115] consiste à faire fonctionner plusieurs systèmes d'exploitation, appelés systèmes invités, au dessus du système d'exploitation de ce serveur appelé serveur hôte, serveur physique ou encore machine hôte (figure 2.10).

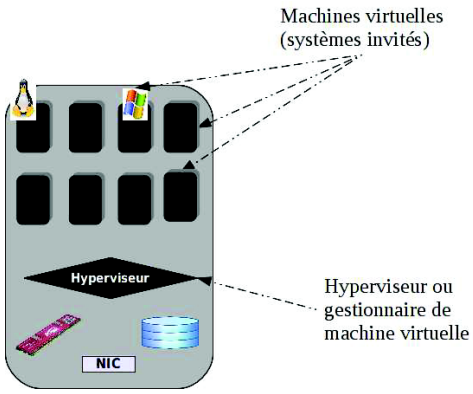

FIGURE 2.10 – Principe de la virtualisation et de la consolidation : plusieurs machines virtuelles sont exécutées simultanément sur un nœud plysique.

Les différents systèmes virtuels – aussi appelés instances virtuelles, machines virtuelles, serveurs virtuels, ou encore systèmes invités – se partagent les ressources physiques du serveur (mémoire, CPU, disque, réseau...), mais la virtualisation doit satisfaire une propriété essentielle : l'*isolation* entre eux. Les serveurs virtuels doivent en effet être isolés mutuellement l'un de l'autre. Autrement dit, aucune des instances virtuelles ne doit pouvoir accéder directement ou indirectement aux ressources réservées à une autre instance.

La virtualisation d'un serveur nécessite un outil spécifique, à l'instar de KVM [85], Microsoft Hyper-V[73], VMware ESXi/ESX (de la suite VMware vSphere [124]) ou Xen [37], pour gérer le cycle de vie des machines virtuelles. Appelé hyperviseur ou un gestionnaire de machine virtuelle est un module logiciel autonome ou intégré à un système d'exploitation traditionnel (comme Windows ou Linux) pour fournir des fonctionnalités de création et de manipulation de machines virtuelles [3].

La gestion de cycle de vie d'une machine virtuelle nécessite des fonctionnalités de :

- création d'une machine virtuelle ;
- démarrage d'une machine virtuelle ;
- suspension d'une machine virtuelle ;
- suppression d'une machine virtuelle ;
- redémarrage d'une machine virtuelle suspendu ;
- arrêt d'une machine virtuelle ;
- migration d'une machine virtuelle d'un serveur à l'autre via le réseau ;

2.3.3.2 Techniques de virtualisation

Nous distinguons quatre grandes techniques : la virtualisation complète, la paravirtualisation, la virtualisation assistée par le matériel et le confinement (ou *sandboxing* en anglais).

Virtualisation complète

Suivant cette technique, le matériel présenté à la machine virtuelle est émulé par translation de code [123]. Les instructions niveau utilisateur sont directement exécutées sur le processeur. Les instructions de bas niveau qui sont notamment liées au matériel sont interceptées et remplacées par une suite d'instructions non-privilégiées pouvant s'exécuter à partir d'une machine virtuelle C'est la translation de code. Voir l'exemple de la figure 2.11 qui illustre le fonctionnement sur une architecture $x86$).

La virtualisation complète ne nécessite aucune modification du système d'exploitation de la machine virtuelle, mais entraine des dégradations de performances importantes.

Paravirtualisation

Dans l'approche paravirtualisée, les instructions qui ne peuvent être directement exécutées à partir d'une machine invitée sont remplacées par des appels systèmes spécifiques

3. Pour information, selon que l'hyperviseur soit un module logiciel autonome ou non par rapport à un système d'exploitation, on parlera d'hyperviseur natif (ex. Xen et VMware ESXi) ou d'hyperviseur hébergé (ex. KVM et Microsoft Hyper-V).

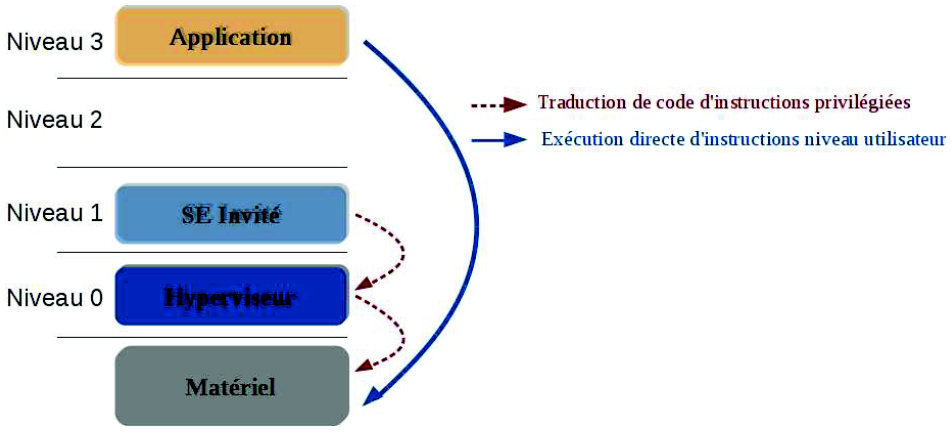

FIGURE 2.11 – Principe de la virtualisation complète sur une architecture $x86$.

(*hypercalls*, sur une architecture $x86$), qui sont interceptées par l'hyperviseur qui se charge ensuite de les traiter. Sur une architecture $x86$, la machine invitée est exécutée avec des privilèges élevés (au niveau 0, voir figure 2.12), comme un système d'exploitation traditionnel.

La paravirtualisation permet d'améliorer significativement les performances en réduisant les surcouts dûs à l'émulation des instructions non-virtualisables. De plus, les systèmes d'exploitation paravirtualisés ont l'avantage d'être légers, vu que seules les instructions essentielles sont virtualisées ; les autres (non-virtualisables) étant implémentées au sein de l'hyperviseur.

La mise en œuvre de la paravirtualisation est cependant contrainte, car nécessitant la modification du système d'exploitation de la machine invitée, afin qu'il puisse supporter les *hypercalls*. Cette modification, qui est spécifique à chaque hyperviseur, est couteuse. Par conséquent, il est courant que plusieurs systèmes d'exploitation majeurs ne soient compatibles avec un hyperviseur paravirtualisé donné : Microsoft Windows, par exemple, n'est pas compatible avec Xen [37] en mode paravirtualisé.

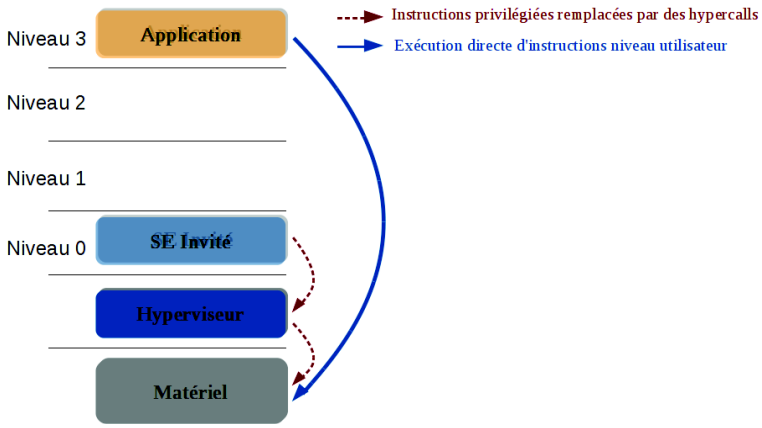

FIGURE 2.12 – Principe de la paravirtualisation sur une architecture $x86$.

Virtualisation assistée par le matériel

La virtualisation assistée par le matériel consiste à créer un nouveau niveau d'exécution privilégié au sein du processeur. Sur une architecture $x86$, par exemple, l'hyperviseur s'exécute au niveau 0. Les appels d'instructions non exécutables au sein d'une machine virtuelle génèrent automatiquement des traps qui sont interceptées et prises en charge par l'hyperviseur (figure 2.13).

Cette approche vise à éviter la modification de code nécessaire avec la paravirtualisation, tout en préservant de bonnes performances.

Elle a deux limitations majeures : (i) en pratique, lorsque plusieurs machines invitées fonctionnant avec des systèmes d'exploitation entièrement non-modifiés s'exécutent sur un même serveur, elles induisent un nombre important de traps dont la prise en charge entraine des surcouts de performances importants [95]. De plus, (ii) la virtualisation assistée par le matériel est contrainte par le matériel qui doit être spécifiquement développé pour : seules les dernières générations de processeurs supportant notamment les technologies Intel-VT [76] ou AMD-V [76] sont compatibles.

Il est possible de combiner la paravirtualisation et la virtualisation assistée par le matériel pour optimiser encore plus les performances des machines virtuelles : c'est la virtualisation hybride (*hybrid virtualization* [95]). Elle a été initialement proposée pour contourner le problème de génération excessif de traps lorsqu'on exécute un système non-modifié avec la virtualisation assistée par le matériel.

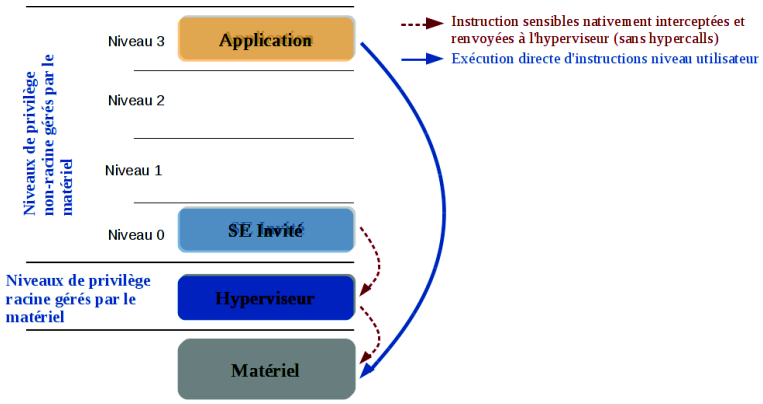

FIGURE 2.13 – Principe de la virtualisation assistée par le matériel sur une architecture $x86$.

Confinement d'application

Le confinement d'application consiste à créer des espaces restreints, appelés conteneurs ou zones, dans lesquels des applications sont exécutées comme si le conteneur les hébergeant est un système d'exploitation à part entière. On distingue deux principales techniques de confinement : l'isolation et l'exécution de noyaux systèmes en espace utilisateur :

Isolation : L'isolation consiste à créer un conteneur qui enveloppe l'exécution d'un ou de plusieurs applications de manière à éviter les interactions de ces applications avec le monde extérieur au conteneur.

Le confinement d'application est solution de virtualisation performante car induisant de faibles surcouts de virtualisation. Sa mise en œuvre repose sur des outils tels que Linux-VServer [16], OpenVZ [23], BSD Jail [3], chroot [18], les Containers Solaris [27] ou encore cpusets [7]. Cependant, les environnements ainsi virtualisés ne sont pas complètement isolés. De plus, l'isolation est dépendante du système d'exploitation et n'est actuellement supportée qu'en environnement Linux ou UNIX.

Système d'exploitation en mode utilisateur : cette technique de virtualisation, quant à elle, consiste à faire tourner un noyau de système d'exploitation en espace utilisateur (ex. User Mode Linux [28], coLinux [6], etc.). Ce noyau s'exécute comme une application en espace utilisateur du système hôte. Le noyau en mode utilisateur a un espace propre dans lequel il contrôle des applications.

Dépendante du système d'exploitation, cette technique de virtualisation est aussi très peu performante à cause de l'empilement des deux noyaux.

2.3.3.3 Virtualisation d'une infrastructure distribuée

La virtualisation d'une infrastructure distribuée consiste à allouer et gérer les ressources des serveurs constituant l'infrastructure via des machines virtuelles. Visant à tirer partie de la flexibilité qu'apporte la virtualisation, elle est usuellement utilisée pour atteindre divers objectifs :

- Fournir des services à la demande et de manière élastique dans des nuages de calcul ;
- Gérer les ressources de calcul d'une organisation de manière centralisée, tout en facilitant l'accès via une allocation à la demande et éventuellement élastique (nuage privé) ;
- Faciliter le déploiement des applications, notamment pour favoriser ou en faciliter la portabilité, l'automatisation et le redéploiement des systèmes ;

La virtualisation d'une infrastructure distribuée nécessite un outil logiciel spécifique appelé gestionnaire d'infrastructure virtuelle (*Virtual Infrastructure Manager*), pour créer et manipuler les machines virtuelles sur l'ensemble des serveurs de l'infrastructure. Un gestionnaire d'infrastructure virtuelle (comme ECP [9], Eucalyptus [98], Nimbus [81], OpenNebula [116], OpenStack [22] et VMware vSphere [124]) ne fournit pas lui-même le support de virtualisation : il repose sur des hyperviseurs déployés sur chaque serveur.

Un gestionnaire d'infrastructure virtuelle doit en principe répondre à un certain nombre d'exigences de fonctionnalités [116] :

- Fournir une vue uniforme et homogène des ressources virtualisées, indépendamment de la technologie de virtualisation sous-jacente. Il doit supporter le plus grand nombre d'hyperviseurs possible (Xen, Hyper-V, KVM...), et être facilement extensible pour supporter d'autres hyperviseurs.
- Gérer le cycle de vie complet d'une machine virtuelle, notamment la configuration automatique du réseau, des périphériques de stockage, ainsi que la création et la personnalisation dynamique de son environnement logiciel.
- Planifier (ordonnancer) l'exécution de machines virtuelles de manière à supporter, par exemple, des mécanismes de priorité, de préemption, d'utilisation efficace de ressources, etc. La stratégie de placement de machines virtuelles sur les serveurs phy-

siques doit être souple et configurable (personnalisable) de manière à être facilement modifiée ou adaptée pour atteindre des objectifs spécifiques d'une organisation (ex. haute disponibilité, économie d'énergie par consolidation des serveurs et équilibrage de la charge des serveurs).
- S'adapter facilement au changement ou redimensionnement de l'infrastructure (augmentation ou retrait de ressources en cas de fluctuation ou baisse de charge) ;
- Supporter un fonctionnement en haute disponibilité, pour répondre notamment à des problématiques de résilience ou de tolérance aux pannes, inévitables dans les systèmes informatiques ;
- Passer à l'échelle : l'outil doit gérer de très grandes infrastructures distribuées. La taille d'une plateforme de calcul atteint facilement plusieurs milliers de serveurs et de machines virtuelles à gérer. Par exemple, en 2006, Intel gérait un parc virtualisé reposant sur plus de $18,000$ serveurs physiques repartis à travers le monde [74].

2.3.3.4 Étude de technologies de virtualisation

Nous présenterons des exemples d'hyperviseurs et de gestionnaires d'infrastructure virtuelle. Nous évaluerons chacune de ces classes de technologies à partir de critères que nous préciserons à chaque fois. De manière générale, le critère cout sera considéré en prenant en compte le type de licence logiciel (gratuit ou commercial), le support (commercial ou communauté), etc.

Hyperviseurs

Il existe un nombre important d'hyperviseurs à l'instar de KVM, Hyper-V, VMware ESXi et Xen que nous avons cités précédemment. Il est difficile de faire une liste exhaustive, ainsi face à cette pléthore d'outils qui offrent plus ou moins les différentes fonctionnalités de base de création et de manipulation de machines virtuelles, nous proposons ci-après des critères supplémentaires pour les évaluer :

Les performances. Elles concernent en particulier les surcouts de performances induits par la virtualisation, le meilleur hyperviseur étant celui ayant les plus faibles surcouts. Remarquons que les surcouts de performances sont souvent étroitement liés à l'approche de virtualisation adoptée (virtualisation complète, paravirtualisation ou virtualisation assistée par le matérielle).

Le support de Virtio [110], qui contribue à réduire considérablement les dégradations de performances liées aux opérations d'entrée-sortie (lecture et écriture disque par exemple). C'est en effet un ensemble d'interfaces qui permettent de réaliser des entrées-sorties performantes (suivant une logique paravirtualisée) au sein de machines virtuelles.

Interopérabilité. Elle vise à évaluer le risque d'enfermement (*lock-in*) avec un hyperviseur, et prend en compte les possibilités d'intégration avec d'autre outils de virtualisation (les gestionnaires de grappe virtuelle notamment), de migration des données ou images de machines virtuelles vers d'autres hyperviseurs, la cohabitation, etc.

L'ouverture des codes, des standards de développement ou des interfaces est un facteur essentiel de l'interopérabilité. D'une part, cette ouverture favorise la compréhension ou la maitrise de l'outil, et d'autre part, l'existence de standards favorise et facilite le développement d'outils tiers compatibles.

Pour faciliter l'interopérabilité des hyperviseurs avec les autres outils de virtualisation, la librairie Libvirt [86]est imposée comme un mécanisme d'unification des interfaces des différents hyperviseurs. Visant à masquer les spécificités des différents hyperviseurs, c'est une suite de logiciels qui fournit des interfaces de programmation en C et des programmes en ligne de commande pour configurer et manipuler les machines virtuelles, de manière indépendante à l'hyperviseur.

Par ailleurs, la compatibilité des images avec le format OVF (*Open Virtualization Format* [102]) est un élément majeur contribuant à l'interopérabilité. Pour information, OVF est un format d'empaquetage d'applications et d'images de machine virtuelle portable sur tous les hyperviseurs compatibles.

Nous étudierons les hyperviseurs suivants selon à leur popularité : Hyper-V, KVM, Xen, XenServer, XCP [4] et VMware ESXi. Nous avons expressément exclu les outils de virtualisation de poste de travail, à l'instar de VirtualBox [5], qui ne sont pas destinés à des infrastructures de calcul. Une synthèse de l'étude est présentée dans le tableau 1 en annexe.

D'un point de vue fonctionnel, ces outils sont très peu différentiables par rapport aux fonctionnalités de gestion du cycle de vie d'une machine virtuelle. De plus, ils sont compatibles avec Libvirt, supportent quasiment tous la paravirtualisation, la virtualisation complète et la virtualisation assistée par le matériel, à part KVM qui ne supporte pas la paravirtualisation en mode natif (nécessite Virtio). Peu d'hyperviseurs, à l'instar des hyperviseurs VMware, supportent actuellement le format OVF, certains outils tels que XenServer fournissent plutôt des outils de conversion à partir du format OVF.

Concernant les performances de ces technologies, les études [62, 125, 122, 128, 37] traitent du sujet. Cependant, elles sont obsolètes (datant de plusieurs années) et ne sont donc plus pertinentes. Ces différentes technologies ont notamment significativement évoluées avec de nouvelles versions de plus en plus optimisées. En outre, aucun des résultats ne prend à la fois en compte toutes ces différentes technologies.

D'un point de vue cout, ces outils se différencient sur plusieurs aspects : leurs licences logicielles respectives – avec des outils open source (Xen ; XenServer, XCP, KVM) ou propriétaires (Hyper-V et VMware ESXi/ESX). Ils se différencient également par l'existence ou non de support. KVM n'a pas de support commercial, tandis que XenServer, VMware et Hyper-V n'ont pas de support communauté.

Gestionnaires d'infrastructure distribuée virtualisée

Nous étudierons Enomaly ECP [9], Eucalyptus [98], Nimbus [81], OpenNebula [116], OpenStack [22] et VMware vSphere [124], six outils majeurs de gestion d'un centre de

4. XenServer et XCP (*Xen Cloud Platform*) sont des dérivées de Xen offrant plus de facilités d'administration, voir http ://xen.org/products/.
5. https ://virtualbox.org/

données virtualisé. Nous avons expressément exclu de cette étude les gestionnaires de ressources des plateformes de nuages publics dont l'implémentation et la documentation sont fermées.

Dans notre étude nous mettons un accent les aspects suivants :

Adaptation/personnalisation et extensibilité : Un gestionnaire d'infrastructure virtuelle doit fournir des mécanismes pour faciliter son intégration, son extension, son adaptation pour de nouveaux besoins. En particulier, les auteurs de [116] suggèrent que le gestionnaire d'infrastructure virtuelle fournisse des interfaces ou APIs standardisées pour sa manipulation via un logiciel tiers. De telles interfaces visent à dissocier le code des outils d'intégration tiers de celui du gestionnaire d'infrastructure, tout en bénéficiant des fonctionnalités fournies par ce dernier. Grâce à cette dissociation, on évite en particulier de gérer les spécificités (du code) du logiciel sous-jacent – avec la complexité qui l'accompagne.

Interopérabilité : Les interfaces d'interaction avec le gestionnaire d'infrastructure doivent être standardisées.

Ce qui n'est pas réellement le cas actuellement. Les interfaces d'Amazon EC2 se sont de facto imposées comme « standard » vu que la majorité de nouveaux outils s'est largement inspiré [9, 98, 81, 116, 22]. Ce sont des interfaces de type service Web reposant sur les protocoles SOAP [96] et REST [64]. En l'absence d'un réel standard, on a vu apparaitre de nouvelles interfaces et APIs spécifiques à chaque outil. C'est par exemple le cas d'OpenNebula avec ses APIs XML-RPC [30].

Même si des efforts de standardisation ont été entrepris, avec notamment le protocole OCCI (*Open Cloud Computing Interface*[20]), cette diversité d'interfaces pose un problème de compatibilité, et donc d'interopérabilité entre ces différents outils.

Ordonnancement et réordonnancement dynamique de machines virtuelles : Dans une infrastructure virtualisée, l'ordonnancement permet de déterminer quand et où (sur quel nœud) placer (déployer et exécuter) une machine virtuelle. Le réordonnancement a, quant à lui, pour but de replacer automatiquement les machines sur les nœuds physiques d'une infrastructure à partir de données collectées dynamiquement sur l'infrastructure (*monitoring*). Ce replacement permet d'atteindre divers objectifs comme l'économie d'énergie ou l'équilibrage de charge.

Les questions d'ordonnancement dans les plateformes de nuage public traditionnelles, à l'instar d'Amazon EC2 ou Go-Grid, ne sont pas réellement pas connues du grand public. A priori, à cause des contraintes d'accès à la demande aux ressources, cela se résumerait à la question de placement des machines virtuelles sur des nœuds physiques : toute nouvelle demande de ressources doit être immédiatement satisfaite en provisionnant les machines virtuelles nécessaires. Au pire des cas, si la demande ne pouvait être satisfaite, elle serait rejetée.

Nous n'en savons cependant pas plus sur la questions de replacement des machines virtuelles qui semble a priori avoir un intérêt. Si, par exemple, les machines virtuelles des clients sont dispersées sur les serveurs physiques, il serait peut-être nécessaire de les replacer pour optimiser l'utilisation des ressources. En particulier, dans des sys-

tèmes autogérés ou auto-adaptables [87, 131], le placement des machines doit souvent être ajusté au fil du temps pour répondre notamment à des problématiques d'économie d'énergie ou de résilience aux fautes.

De manière générale, la considération d'allocation immédiate de ressources n'est pourtant pas réaliste dans bien des cas de gestion d'infrastructure virtualisée. Considérons, par exemple, le cas d'une entreprise qui virtualise ses serveurs pour fournir des ressources de test pour ses applications ou directions (comme chez Intel [74]). Il serait envisageable et réaliste d'allouer les ressources aux machines virtuelles via un mécanisme de file d'attente, qui prend notamment en compte des priorités entre les différentes applications utilisant l'infrastructure.

Pour encore plus apprécier les problématiques d'ordonnancement et de réordonnancement de machines virtuelles sur une infrastructure distribuée, le lecteur pourra par exemple se référer à des projets tels que Haizea [42] , Entropy [72] ou Green Cloud Scheduler [59].

Hyperviseurs supportés : un gestionnaire d'infrastructure virtuelle doit être indépendant de l'hyperviseur sous-jacent et supporter plusieurs hyperviseurs de manière à éviter l'enfermement par rapport à une technologie de virtualisation donnée. Il doit en plus être facilement extensible pour supporter de nouveaux hyperviseurs.

Eucalyptus [98] est une suite logiciel conçue pour mettre rapidement en place un nuage de calcul privé IaaS calqué Amazon EC2. L'objectif était clair de simplifier la migration de EC2 vers une plateforme Eucalyptus : les interfaces de Walrus, le service de stockage de données d'Eucalyptus, sont compatibles à celles d'Amazon S3 (*Simple Storage Service* [1]) fournissant le même service pour EC2 ; tandis que les interfaces Eucalyptus EBS (*Elastic Block Storage*) fournissant des volumes de stockage en mode bloc aux machines virtuelles sont identiques à celles d'Amazon EBS

Ayant été calquée sur Amazon EC2, la conception d'Eucalyptus ne semble pas avoir eu de réelles considérations concernant l'adaptation et les possibilités d'extension. Les possibilités offertes via ses API (compatibles EC2) sont très limitées. Par exemple, il n'y a aucun moyen d'influencer le placement d'une machine virtuelle sur un nœud physique. L'ajout ou la modification de la stratégie de placement, par exemple, nécessiterait des modifications au niveau du code ; ce qui est typiquement complexe et comporte un risque important de générer des bugs ou effets de bord.

Nimbus : Comme Eucalyptus, Nimbus [81] est un outil pour déployer des nuages IaaS calqué sur Amazon EC2. Open source et héritage du projet Globus [11], Nimbus cible la communauté scientifique en cherchant à offrir des fonctionnalités avancées comme une authentification renforcée (en utilisant des certificats par exemple) et des mécanismes de gestion de ressources avancés.

Nimbus supporte les hyperviseurs Xen et KVM.

Comme dans Eucalyptus, le placement de machine virtuelle est immédiat basé sur une stratégie gloutonne ou tourniquai. Il apporte cependant un ajout en offrant un mode de fonctionnement où des machines virtuelles sont exécutées en mode best-effort (préemptables)

[91]. Pour information, ce mode de fonctionnement (*Pilot mode* dans le jargon Nimbus) vise un couplage avec un gestionnaire de ressources volatile externe (ex. Condor [89]) afin de maximiser l'utilisation des ressources d'une infrastructure Nimbus. Ce mode de fonctionnement est similaire à celui du projet Cloud Scheduler [5] qui vise à proposer un outil générique pouvant supporter divers gestionnaires d'infrastructure virtuelle.

La personnalisation de Nimbus repose sur un mécanisme de plugin par injection de dépendances [105]. Nimbus étant basé sur la framework Spring [6], un tel plugin est donc dépendant du langage Java qui sous-tend Spring. Ses APIs d'accès et de manipulation sont compatibles à celles d'Amazon EC2 ainsi qu'à celles d'Amazon S3.

ECP (*Elastic Computing Platform*) [9] est un outil commercial visant à apporter une solution intégrée clé en main pour déployer des nuages de calcul IaaS. Il est également calqué sur le modèle Amazon EC2, mais ne fournit pas d'interfaces compatibles avec ce dernier.

Étant une solution propriétaire et de conception non ouvertement documentée, nous n'avons pas de connaissance concernant ses possibilités de personnalisation et de placement de machines virtuelles. D'après les informations disponible sur son site Internet, il supporterait les hyperviseurs Xen, KVM et VMware.

OpenNebula [116] est disponible en versions gratuite et commerciale (étendue). Cet outil est présenté comme étant modulaire, flexible, adaptable, performant et scalable grâce à son architecture initialement conçue pour satisfaire les besoins du projet RESERVOIR [7] dont il est issu.

OpenNebula prend en charge des machines virtuelles Xen (et aussi XenServer et XCP dans la version commerciale), KVM, VMware et Hyper-V.

Le provisionnement de machines virtuelles est géré par un ordonnanceur.

L'ordonnanceur par défaut (mm_sched) gère une file d'attente où les machines virtuelles sont sélectionnées suivant une stratégie FCFS. La sélection du nœud où une machine virtuelle sera placée repose sur des affinités entre la machine virtuelle et le nœud (*matchmaking*). Les paramètres d'affinités sont spécifiés lors de la définition de la machine virtuelle. Ils concernent, par exemple, la quantité de CPU ou de mémoire disponible, le type d'hyperviseur sous-jacent, le nom de la machine, l'adresse MAC, etc. Pour plus de détails, le lecteur intéressé peut consulter la documentation disponible sur le site d'OpenNebula.

La stratégie d'ordonnancement peut être modifiée en remplaçant l'ordonnanceur par défaut. En particulier, Haizea [42] peut être couplé à OpenNebula comme ordonnanceur de machines virtuelles. Haizea est un moteur d'ordonnancement (*meta scheduler*) fournissant des fonctionnalités de gestion de baux de ressources via des machines virtuelles. Il supporte aussi bien des stratégies simples qu'avancées pour la planification des baux (comme le FCFS avec remplissage agressif ou conservatif, le placement immédiat, la réservation par avance, la gestion de priorité et la préemption). Toutefois, ces fonctionnalités ne sont pas toutes prises en charge lorsque Haizea est couplé avec OpenNebula : c'est en particulier le cas de la migration d'une machine suspendue.

6. http ://www.springsource.org/
7. http ://www.reservoir-fp7.eu/

De manière générale, OpenNebula est basé sur une architecture modulaire et adaptable, qui fournit des interfaces et APIs compatibles XML-RPC, OCCI et Amazon EC2. Cependant, contrairement à Eucalyptus, ECP, ou encore Nimbus, son modèle de conception n'est pas une copie du modèle d'Amazon EC2. Le niveau de personnalisation est assez fin grâce à son architecture et ses outils de *contextualisation* [113] : quasiment toutes ses fonctionnalités essentielles sont accessibles et manipulables via ses APIs XML-RPC et OCCI. De plus, OpenNebula fournit divers outils et une documentation assez fournie pour faciliter son extension avec notamment des ajouts de fonctionnalités qui sont basées sur des mécanismes non accessibles par le biais de ses APIs (ex. ajout de support d'un nouveau hyperviseur).

OpenStack [22], est une collection d'outils open source conçue pour le déploiement de nuages IaaS massivement scalables et robustes (infrastructures reposant sur un très grand nombre de machines virtuelles et de très grands volumes de données avec des exigences de scalabilité et redondance). Nouveau venu des outils IaaS étudiés ici, OpenStack est issu des briques du projet Nebula [8] de la NASA et RackSpace Hosting [25].

OpenStack supporte les hyperviseurs Xen, XenServer/XCP, KVM, UML (*User-Mode Linux*), VMware ESXi/ESX et Hyper-V.

Son architecture définit un module de gestion de ressources chargé du placement des machines virtuelles sur les nœuds d'une infrastructure distribuée. Le gestionnaire de ressources par défaut (*nova-schedule*) – qui semble personnalisable via une adaptation appropriée au niveau du code – est assez basique car il ne supporte que des placements immédiats selon trois stratégies au choix :

- *Chance :* le nœud est choisi de manière aléatoire entre les serveurs disponibles.
- *Zone disponible :* Similaire à la politique *chance*, mais le nœud est choisi aléatoirement dans une zone disponible donnée. Une zone dans OpenStack représente un ensemble de serveur.
- *Simple :* le nœud sélectionné est celui ayant la plus faible charge.

Malgré qu'il soit calqué sur Amazon EC2 (ex. Eucalyptus), OpenStack a l'avantage (contrairement à Eucalyptus) d'être basé sur une architecture modulaire visant à faciliter son adaptation.

VMware vSphere [124], est une suite d'outils de virtualisation de centre de données commercialisée par VMware. La suite fournit des outils qui, intégrés ensemble permettent d'agréger et de gérer un ensemble de serveurs virtualisés – via VMware ESXi/ESX notamment. Contrairement à Eucalyptus, Nimbus, OpenNebula, et tous les autres outils cités précédemment, cette suite n'a pas été originellement conçue comme une solution destinée à des nuages IaaS. Anciennement *VMware Infrastructure*, elle a été à l'origine conçue pour optimiser l'utilisation de ressources grâce à la virtualisation. L'offre a simplement évolué pour s'adapter à la demande.

Le placement de machines virtuelles dans une infrastructure vSphere est géré manuellement, automatiquement ou sémi-automatiquement grâce à vSphere DRS (*Distributed Re-*

8. http ://nebula.nasa.gov/

source Scheduler). Conçu à l'origine comme outil d'équilibrage de charge, DRS gère le placement initial des machines virtuelles.

La politique de placement est rudimentaire, ne supportant que de simples paramètres liés notamment aux affinités entre des machines virtuelles. Par exemple, on peut suggérer ou exclure la cohabitation de certaines machines virtuelles sur un même nœud, pour répondre à des exigences de performance ou de haute disponibilité par exemple.

Une synthèse de l'étude est présentée dans le tableau 2 en annexe.

2.3.3.5 Les plus et les moins de la virtualisation

La virtualisation facilite la configuration et la reconfiguration des environnements de calcul. En effet la manipulation d'une machine virtuelle, c'est-à-dire leur création, déploiement, destruction, sauvegarde, etc., est plus simple et plus rapide que la manipulation d'un système réel. De plus, l'image d'une machine virtuelle peut être aisément sauvegardée pour un déploiement ou redéploiement ultérieur. Enfin, l'image d'une machine virtuelle peut être dupliquée facilement et simplement pour créer plusieurs machines virtuelles.

L'utilisation de machines virtuelles favorise par ailleurs la mise en place des techniques telles que la consolidation des ressources (en exécutant simultanément plusieurs machines virtuelles sur un nœud physique, figure 2.10), l'hébergement mutualisé, des environnements de calcul isolés (*sandbox* ou boucliers contre des programmes malicieux), la portabilité et la migration des applications, etc.

La virtualisation de serveur a néanmoins des contraintes, liées notamment aux surcouts de performances des machines virtuelles [125, 128, 122].

Toutefois, il existe des solutions d'optimisation qui tentent de répondre à cette problématique en contournant, tant que possible, l'émulation des instructions non-virtualisables.

Nous pensons notamment aux solutions de virtualisation assistée par le matériel telles qu'Intel-VT [76] et AMD-V [31], où les instructions non-virtualisables génèrent des *traps* qui sont interceptées et traitées par le matériel. La paravirtualisation, qui est une approche logicielle, où les instructions non-virtualisables sont remplacées par des appels systèmes spécifiques (*hypercalls*) qui sont traités par l'hyperviseur. Enfin d'autres solutions logicielles telles que les APIs Virtio [110], PCI Passthrough [79] ou de manière générale les mécanismes dits d'*hypervisor-bypassing* [90], qui permettent de réaliser des entrées/sorties via des appels systèmes spécifiques qui outrepassent la couche de virtualisation.

2.4 Synthèse

Dans ce chapitre, nous avons présenté un état de l'art général sur le calcul dans le nuage et le calcul intensif. Un regard particulier a été porté sur différents aspects d'exploitation d'une plateforme de calcul (gestion de ressources, l'ordonnancement de tâches et la virtualisation de ressources).

Pour compléter cet état de l'art, nous aborderons au chapitre suivant la question de la mutualisation avec les problématiques de gestion de ressources qu'elle soulève.

De l'intérêt d'une mutualisation de ressources aux problématiques d'exploitation 3

Dans ce chapitre, nous présenterons d'abord l'intérêt de la mutualisation de ressources. Nous analysons dans la section 3.2 les problématiques de gestion de ressources qu'elle soulève. Ensuite, nous montrerons dans la section 3.3 que les outils existants ne sont pas adaptés. Le chapitre se terminera par une discussion qui montrera la nécessité de définir de nouvelles solutions pour répondre aux questions soulevées par cette mutualisation.

3.1 Pourquoi la mutualisation de ressources ?

Les infrastructures actuelles comportent plusieurs dizaines, voire plusieurs milliers, de serveurs. Mais elles sont onéreuses à cause des charges d'acquisition et d'exploitation. Nous pensons par exemple aux couts des serveurs, des réseaux d'interconnexion, des systèmes de refroidissement ou à la consommation énergétique.

Malgré ce caractère onéreux, des études montrent que toute la puissance de calcul délivrée est souvent sous exploitée [94, 34, 91] : les moyennes d'utilisation varient entre 5 à 20% [34]. Pire, les solutions proposées jusqu'à l'heure ne permettraient d'atteindre qu'à peine les 50% de taux d'utilisation [57].

Une sous-utilisation de ressources a diverses causes dont les principales sont :

- Une demande en dents-de-scie : elle se symbolise par une demande en puissance de calcul qui augmente ou baisse au fil du temps. Pour illustration, ce phénomène est observable sur la figure 3.1 où nous avons montré le nombre de processeurs réservés par les utilisateurs dans Grid'5000 entre les semaines 47 et 50 de 2011 [1] (mi-novembre et mi-décembre). Nous observons en effet qu'il y a des périodes où les ressources sont excessivement sollicitées (ex. entre les semaines 48 et 49) et des périodes de faibles demandes (ex. entre les semaines 49 et 50).
  Dans un contexte de nuage de calcul, cette question serait encore plus criante dans une phase de lancement de service (les premiers mois ou années de fonctionnement), car la demande est en général encore faible.
- Une fragmentation de ressources : imaginons, par exemple, qu'un nœud multicœurs soit dédié à l'exécution d'une application séquentielle. Dans une telle situation, seul

1. Source : `https://helpdesk.grid5000.fr/ganglia/`

un cœur pourra être réellement utilisé et conduira à une sous-utilisation de ressources. Ce qui justifierait le fait que, dans le cas de Grid'5000 où les nœuds sont pour la plus part composés de quatre ou huit cœurs, la charge moyenne des nœuds sur 15, 5 et 1 minute(s) étaient respectivement de 4%, 4% et 5% dans la même période.

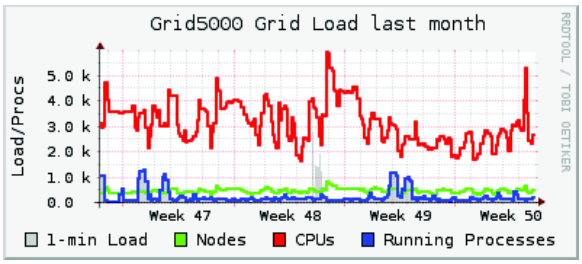

FIGURE 3.1 – Charge de Grid'5000 entre mi-novembre et mi-décembre 2011.

Pour une grande entreprise disposant de budget de fonctionnement conséquent, le cout d'une infrastructure de calcul est en général relativement faible par rapport au budget de fonctionnement. Mais pour une petite ou moyenne entreprise/industrie dont ce budget est limité, cela est plus contraint de sorte que disposer et exploiter une plateforme de calcul importante semble hors de portée.

Avec l'apparition du calcul dans le nuage, une alternative à une plateforme de calcul interne à une organisation consiste à utiliser à la demande des ressources via un nuage tel que Amazon EC2 [1] ou Raskspace Hosting [25]. Cette solution a l'avantage de permettre de payer les ressources à l'utilisation et d'éviter une sous-utilisation des ressources. Cependant, elle présente des inconvénients car la location d'une capacité de calcul importante sur le long terme peut être très onéreuse [52]. De plus, lorsque les besoins augmentent, une demande d'extension de capacité peut ne pas aboutir : en effet, la notion d'élasticité infinie des capacités fournies par un nuage n'est qu'une illusion basée sur l'hypothèse que la quantité de ressources exigées par les clients d'un nuage n'excédera jamais la limite du nuage. C'est pour ces raisons que nous pensons que la location de ressources via un nuage est plutôt adaptée pour répondre à des augmentations ponctuelles de charge sur une infrastructure de calcul interne.

Dans tous les cas, que ce soit une petite entreprise ou une grande organisation disposant d'une plateforme de calcul privée, les ressources doivent être utilisées de manière efficace pour mieux rentabiliser les investissements. La définition des mécanismes de gestion de ressources (comme les politiques d'ordonnancement de tâches, d'affectation de ressources ou de virtualisation) est donc déterminante.

3.2 La mutualisation et ses contraintes de gestion de ressources

Nous décrirons d'abord le principe de la mutualisation et nous nous intéresserons davantage à un cas d'étude qui sera analysé dans la suite de cette thèse.

3.2.1 Principe de la mutualisation

Visant particulièrement des petites ou moyennes entreprises, il s'agira pour plusieurs entreprises de mettre ensemble leurs moyens pour acquérir et exploiter une plateforme de calcul commune. La puissance de calcul délivrée devra alors être partagée entre leurs applications et leurs utilisateurs de sorte à tenir compte des investissements de chacune.

Cette façon de procéder présente divers intérêts :

- Dans une phase pilote ou d'exploration, elle permet à une entreprise de minimiser son investissement et donc de limiter les risques.
- Elle permet d'avoir accès à une grande plateforme de calcul (acquise et exploiter grâce aux moyens mutualisés des différentes entreprises).
- Elle laisse toute la liberté (à un niveau système non accessible dans un nuage public) concernant les outils d'exploitation ou des applications que ces entreprises souhaitent déployer.
- Les ressources seront gérées de manière centralisée pour répondre à des objectifs fixés par ces entreprises.

Cette mutualisation est à différencier de la vision des grappes ou des grilles de calcul classiques. En effet, une grappe de calcul appartient en général à une unique organisation et par conséquent, comme dans une approche multitenant, la gestion de ressources est guidée par les seuls objectifs de cette organisation. Les ressources d'une grille, quant à elles, appartiennent à différentes organisations mais les mécanismes de gestion ne sont pas centralisés.

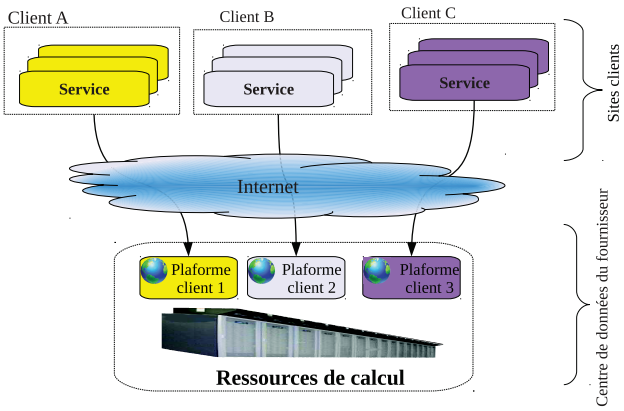

FIGURE 3.2 – Hébergement multitenant.

3.2.2 Cas d'étude et objectifs

Plus spécifiquement, nous considérons dans cette thèse une grappe composée de nœuds multicœurs (SMP ou NUMA) acquise et exploitée par un ensemble limité d'entreprises pour déployer et fournir des services logiciels (SaaS) à leurs clients.

Les services s'appuient sur des applications de calcul intensif et peuvent durer plusieurs heures. Pour une telle plateforme, les clients soumettent leurs requêtes via Internet comme sur la figure 3.3. Cette figure illustre une situation avec trois entreprises qui proposent des services reposant sur trois applications App_1, App_2 et App_3.

Dans ce contexte, une requête est une demande d'utilisation d'une application par un client avec un jeu de données. L'exécution de requête appelée tâche, consiste à traiter le jeu de données avec l'application concernée. A la fin du traitement un résultat est retourné au client.

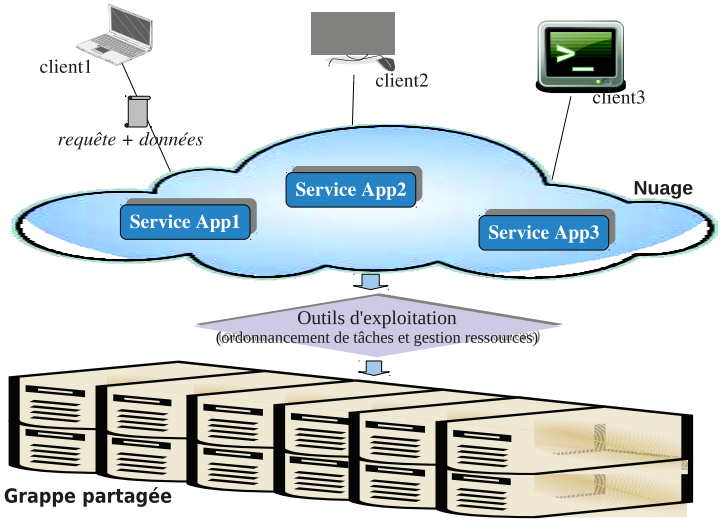

FIGURE 3.3 – Plateforme de nuage avec des ressources partagées

Nous visons à atteindre les objectifs de gestion de ressources suivants :

Gestion dynamique de tâches avec contraintes priorités : Après qu'un client ait soumis une requête, l'allocation et l'affectation de ressources pour l'exécution de la tâche associée doivent être transparentes pour le client. De même, les ressources allouées pour l'exécution doivent être automatiquement libérées à la fin de l'exécution de manière transparente. De manière générale, toute la complexité liée à la gestion du partage, de l'allocation ou de l'assignation de ressources doit être cachée au client.

Concernant la gestion de priorités, chaque entreprise ou fournisseur de service pourrait différencier les tâches de ses clients selon des critères comme la fidélité (clients réguliers, clients ponctuels ou partenaires). Par exemple, les tâches d'un client auront une priorité plus ou moins forte en fonction de sa catégorie. En outre, un fournisseur pourrait aussi exploiter des ressources inutilisées de la plateforme pour traiter des tâches internes (ex. tests de validation lors des phases de développement de son application). Ces tâches devront alors être traitées avec une très faible priorité, notamment en mode best-effort, par rapport à une tâche cliente. Une ou plusieurs tâches best-effort peuvent être préemptées (suspendues ou arrêtées) pour libérer des ressources nécessaires à l'exécution d'une tâche cliente.

Partager et utiliser efficacement les ressources : Étant donné que chaque entreprise investit sur la grappe, les ressources de la grappe doivent être partagées entre leurs applications de manière proportionnelle à leurs investissements respectifs : chaque service doit pouvoir utiliser un certain ratio de ressources qui soit proportionnel à l'investissement du fournisseur du service.

Pour optimiser l'utilisation de la grappe, les ressources inutilisées doivent au mieux

être utilisées pour traiter des tâches de n'importe quel fournisseur. De plus, on pourra supporter l'exécution de plusieurs tâches notamment lorsque la capacité (nombre de cœurs et la mémoire vive disponibles) d'un nœud le permet. Mais nous devons garantir dans ce dernier cas que chacune des tâches s'exécutant sur le nœud soit cloisonnée vis-à-vis des autres. C'est-à-dire que chacune d'elles doit avoir accès de manière indépendante et exclusive aux ressources affectées pour son exécution.

3.3 A propos de l'existant

Dans cette section, nous préciserons d'abord notre positionnement dans l'écosystème des nuages de calcul. Ensuite, par rapport aux problématiques à traiter, nous montrerons les limites des mécanismes existants concernant notamment le partage de ressources et la gestion de tâches.

3.3.1 Positionnement dans l'écosystème des nuages de calcul

La mutualisation est à distinguer de l'hébergement *multitenant* (*Multitenancy* en anglais [127, 36]). La logique multitenant, comme illustré sur la figure 3.2, se définit comme le fait d'utiliser une même infrastructure pour fournir un même service à plusieurs clients. En d'autres termes, les ressources d'une infrastructure multitenant appartiennent à une seule organisation (celle qui fournit le service). Par conséquent, les objectifs de gestion de ressources sont fixés par cette organisation et ne visent que ses seuls intérêts. Ce qui n'est pas le cas dans un contexte de mutualisation.

Un modèle SaaS reposant sur des applications de calcul intensif se démarque également des nuages SaaS existants comme Salesforce [26] ou Office365 [21]. En effet, contrairement aux applications de bureautique ou collaboratives que l'on retrouve dans ces derniers, les applications de calcul considérées ne nécessitent pas d'interaction avec les clients. C'est ce qui a en réalité motivé l'objectif de prise en charge et de traitement des tâches de manière transparente pour les clients.

Par ailleurs, les plateformes PaaS du marché, par exemple Google App Engine [13] ou Force.com[127], paraissent peu adaptées pour des services s'appuyant sur des applications de type calcul intensif. De par les technologies qu'elles supportent, ces nuages semblent notamment avoir été particulièrement conçus pour des applications web. En effet, ces technologies sont essentiellement basées sur des langages de programmation tels que Java, Ajax, PHP ou .Net. Les possibilités d'adaptation pour des besoins spécifiques sont de surcroit limitées car, étant issues d'entreprises qui ont des exigences de compétitivité, leurs implémentations sont fermées.

3.3.2 Mécanismes de partage de ressources

Partager proportionnellement les ressources d'une plateforme de calcul tout en maximisant leur utilisation sont deux objectifs conflictuels difficiles à atteindre à la fois dans un contexte de mutualisation [52].

Dans la suite, nous commencerons par étudier une technique de partage simple et intuitive basée sur un découpage statique de ressources. Cela nous permettra de bien identifier les problèmes de partage proportionnel et d'utilisation efficace de ressources que nous avons à traiter. Ensuite, par rapport à ces problèmes, nous analyserons le *Fair-share* (cf. page 29), qui est la technique communément employée par les gestionnaires de ressources existants (ex. Maui [41], LoadLeveler [17], SLURM [97] ou PBS Works [24]).

3.3.2.1 Partage par découpage statique

Dans notre contexte de mutualisation, cette consisterait à découper la plateforme en sous-ensembles de tailles proportionnelles aux investissements des différentes entreprises. Ensuite, le sous-ensemble de ressources d'une entreprise donnée serait assigné pour l'exécution des tâches relatives au service qu'elle propose.

Par exemple, la figure 3.4 présente un cas de découpage où une plateforme composée de sept ressources ($Ri\ i = 1 \ldots 7$), vues comme des nœuds bi-processeurs, est mutualisée entre trois entreprises ($e_i\ i = 1 \ldots 3$) pour supporter leur service, $Appi$, $i = 1 \ldots 3$. Cet exemple suppose que ces entreprises investissent pour au moins utiliser l'équivalent des $2/7$, $2/7$ et $3/7$ des ressources, respectivement. L'exemple assume également que plusieurs tâches peuvent être exécutées simultanément sur un nœud tant que la capacité de calcul fournie par ce dernier le permet. Par exemple, un nœud peut exécuter deux tâches nécessitant un cœur chacune, ou une tâche qui en demande 2. Sur les schémas (*a* et *b*) montrant nos cas d'étude, une tâche est représentée par une boite : la couleur d'une tâche indique le service auquel elle est liée, tandis que la hauteur de la boite indique le nombre (1 ou 2) de cœurs nécessaires pour son exécution.

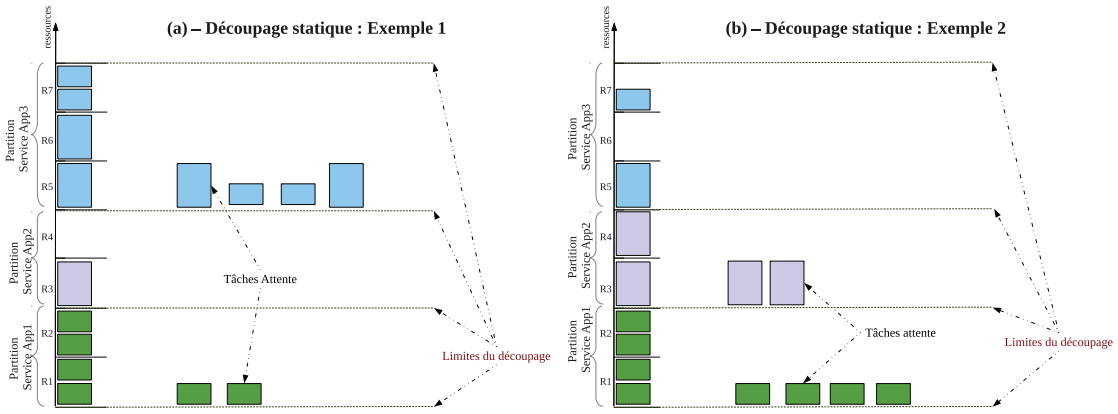

FIGURE 3.4 – Partage de ressources par découpage statique.

Cette solution de partage permet de toujours respecter l'équité ou les proportions d'utilisation de ressources affectées aux tâches de chaque service. Elle parait en outre simple à mettre en œuvre, par exemple grâce à un système de file d'attente multiples comme le montre la figure 3.4.

Par contre, elle présente des problèmes de sous-utilisation de ressources comme nous pouvons le constater :

- Considérons par exemple la configuration *a* (*Exemple 1*). La ressource R_4 affectée au service *App2* (où la demande est faible) reste inutilisée, pourtant des tâches liées aux autres services attendent.
- En observant la configuration *b*, le même phénomène est remarquable puisque les ressources R_6 et R_7 sont inutilisées (ou sous-utilisées) tandis que des tâches des autres services attendent.

Dans tous les deux cas, à cause du découpage strict, des tâches dont l'exécution pourrait créer un dépassement de seuil d'utilisation sont systématiquement mises en attente. Ce qui conduit à une sous-utilisation de l'ensemble des ressources. Il convient donc de faire un compromis entre l'équité et l'utilisation.

3.3.2.2 Partage équitable ou *Fair-share*

La stratégie *fair-share* ou stratégie de partage équitable (cf. section 2.3.2.1, page 29) est une technique de partage de ressources populaire adoptée par beaucoup de gestionnaires de ressources comme Maui [41], LoadLeveler [17], SLURM [97] ou encore PBS Works [24].

Dans un contexte de mutualisation comme le nôtre, elle fonctionne comme suit : lorsqu'une tâche attend (quelque soit le service où elle est liée) tandis qu'il y a des ressources libres, ces dernières seront affectées pour exécuter la première. Sachant que l'équité est assurée en faisant varier à la hausse ou à la baisse les priorités des tâches des services selon que ces derniers ont utilisés plus ou moins de ressources.

Pour illustration, reprenons les exemples précédents en supposant que les configurations *a* ou *b* ont été atteintes en appliquant une stratégie *fair-share*, au lieu d'un découpage statique. Assumons en outre que dans ces différentes configurations, les priorités venaient d'être mises à jour. Nous pouvons alors imaginer des changements de configuration comme sur la figure 3.5 :

- A partir de la configuration *a* (*Exemple 1*), une tâche en attente liée au service *App2* (qui utilise moins de ressources) devrait avoir la plus grande priorité. Alors qu'une tâche liée au service *App1* aurait une forte priorité par rapport à une autre liée au service *App3*.
- A partir de la configuration *b* (*Exemple 2*), une tâche liée au service *App3* aurait la plus forte priorité, tandis que des tâches liées au service *App1* ou au service *App2* auraient les mêmes priorités.

En résumé, ce mécanisme contribue à améliorer le taux d'utilisation de ressource. Mais, concernant le partage de ressources, il peut conduire à des situations d'attente peu convenables dans un contexte de mutualisation.

Par exemple, considérons le changement de configuration à partir de la situation *a* (figure 3.5). Supposons ensuite que les tâches en cours d'exécution nécessitent des durées d'exécution longues. Enfin, assumons qu'après le changement de configuration, une tâche liée au service *App2* est en attente et que son exécution nécessite un cœur. Alors cette tâche devra rester en attente aussi longtemps qu'un cœur ne sera pas libéré (après terminaison d'une ou de plusieurs tâches). Conduisant ainsi à une *longue attente*, qui n'est pas acceptable pour un service qui utilise peu de ressources.

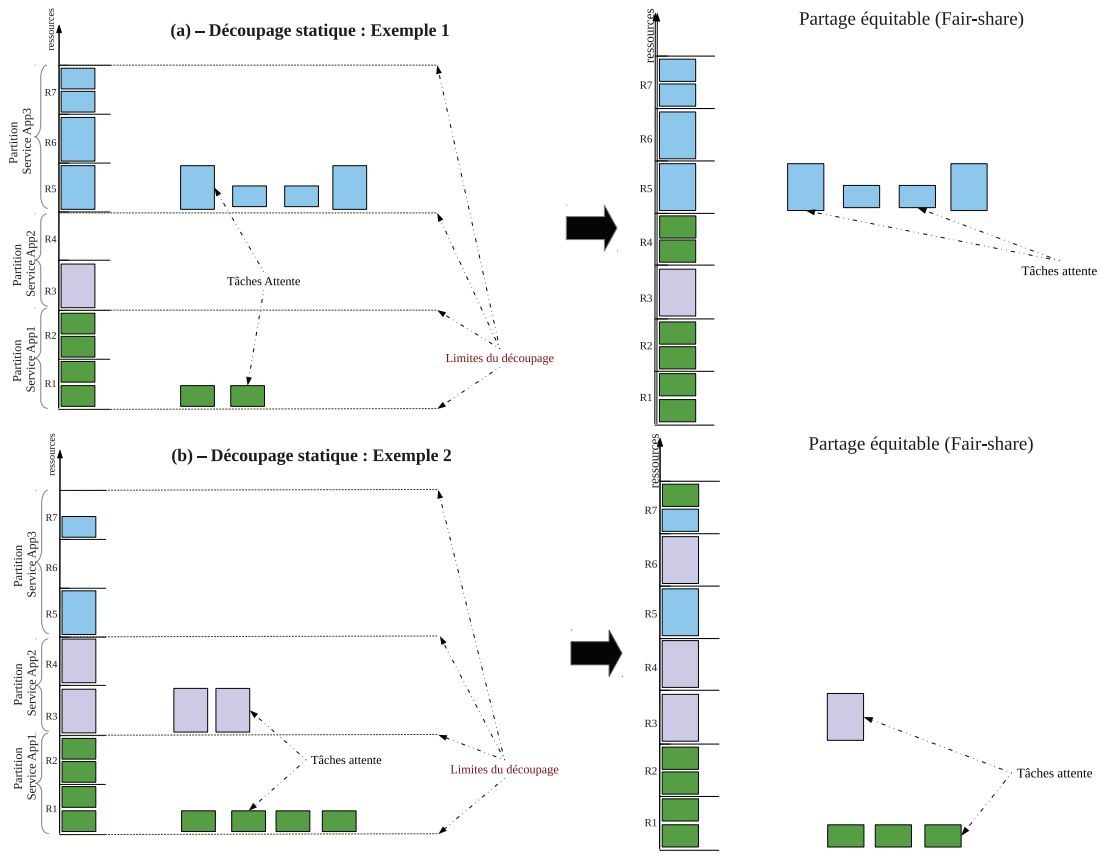

FIGURE 3.5 – Découpage statique vs Partage équitable.

3.3.2.3 Améliorations du partage équitable : préemptions/migrations de tâches

Des migrations ou des préemptions peuvent être envisagées pour atténuer ce problème de longue attente, mais ces solutions soulèvent, elles aussi, d'autres problématiques difficiles à traiter :

Partage équitable avec migration : cela consisterait à migrer dynamiquement des tâches d'un service qui surutilise des ressources sur un sous-ensemble réduit de ressources. Par exemple, prenons la situation dérivée de la configuration *a* de la figure 3.5. Si nous supposons qu'une tâche liée au service *App2* est en attente et nécessite un cœur pour s'exécuter. , alors, il faudrait par exemple migrer une tâche du service *App1* de la ressource R_4 vers la ressource R_1.

Cette solution apporte une flexibilité supplémentaire en garantissant dynamiquement le respect des ratios d'utilisation. Toutefois, une telle migration expose à un risque de surcharge sur le nœud de destination. Ce qui induirait des dégradations de performances. Dans l'exemple choisi, la ressource R_1 se retrouverait à supporter l'exécution de tâches demandant au total trois cœurs : c'est-à-dire plus du nombre de cœurs disponibles sur un nœud.

Partage équitable avec préemption : Au lieu des migrations, il s'agirait de suspendre ou arrêter une ou plusieurs tâches liées à des services qui surutilisent des ressources.

Les questions soulevées dans ce cas sont différentes, mais aussi difficiles à traiter :

– A partir de quel(s) critère(s) déterminer la (ou les) tâche(s) à suspendre, de sorte que cela soit juste vis-à-vis des différents fournisseurs de services ? Par exemple, reprenons la situation dérivée de la configuration *b* (figure 3.5). Si nous supposons alors qu'une tâche liée au service *App3* est en attente et nécessite un cœur pour s'exécuter. Alors il faudrait suspendre une tâche liée soit au service *App1*, soit au service *App2* pour libérer des ressources. D'où un dilemme non évident à résoudre, comment choisir la tâche victime ?
– Quel mécanisme de préemption mettre en place (préemption avec ou sans sauvegarde de l'état du fil d'exécution) ? En effet, il est usuel d'avoir besoin d'un mécanisme de type de sauvegarde/reprise (*checkpointing* [101, 111]) afin d'éviter de perdre des calculs déjà réalisés après une préemption. Mais cette technique a des restrictions, par exemple lorsqu'une application a des fichiers ou des sockets ouvert(e)s.

3.3.3 Environnement d'exécution de tâches

Un autre aspect que nous considérons est lié à la manière d'ordonnancement l'exécution des tâches. Usuellement, une tâche est exécutée à partir d'un environnement logiciel préalablement préparé par un administrateur. Cet environnement peut être déployé sur une machine réelle ou au sein d'une machine virtuelle, chacune des approches présentent des avantages et des inconvénients.

3.3.3.1 Environnement d'exécution sur une machine réelle

Traditionnellement utilisée dans le monde du calcul haute performance, cette manière de procéder permet de bénéficier des performances réelles d'un nœud. Mais elle présente des limites pour assurer l'exécution dynamique de tâches, notamment lorsque les ressources sont partagées. En effet :

Comment changer dynamiquement la configuration logicielle d'un nœud ? Imaginons que, pour être exécutée, des tâches de deux services nécessitent des versions distinctes d'une bibliothèque (ex. *libc++* pour des applications *C++*). Comment dans ce cas assurer qu'un nœud puisse supporter l'exécution de ces tâches ?

Nous savons que, comme OAR [49], certains gestionnaires de ressources offrent la possibilité de redéployer l'environnement logiciel d'un nœud. Toutefois, le temps de déploiement reste assez important. Pour illustration, dans Grid'5000 où la réinstallation repose sur Kadeploy [2], le temps de déploiement d'un système Linux de base (sans outils logiciels spécifiques) se situe autour de 3.5 minutes. Ce qui représente plus de 10% de la durée d'exécution d'une tâche de 30 minutes.

Pour éviter un redéploiement, nous pouvons imaginer affecter à chacun des services un ensemble de nœuds sur lesquels ses tâches seront exécutées. Ce qui reviendrait en réalité à un découpage statique qui, nous l'avions vu, est très peu flexible et conduit à une sous-utilisation de ressources.

2. `http://kadeploy3.gforge.inria.fr/`

Comment assurer la coexistence et le cloisonnement de tâches sur un nœud ? Notons à cet effet que des solutions de type cpusets [7] ou cgroups [4], traditionnellement employées pour le partitionnement de ressources dans le domaine du calcul haute performance, sont très contraintes. En particulier, elles sont dépendantes des systèmes d'exploitation et en outre, leur manipulation est très sensible car nécessitant des privilèges d'administrateur.

3.3.3.2 Environnement d'exécution au sein d'une machine réelle

Avec l'émergence de la virtualisation et des nuages de calcul (publics, privés ou communautaires), les machines virtuelles sont devenues des ressources de calcul courantes. Leur utilisation permet notamment de répondre à des pics de charge ou à optimiser l'utilisation de ressources au sein d'une infrastructure.

Comme expliqué dans [91], par exemple, les machines virtuelles sont typiquement intégrées dans un gestionnaire de ressources classique comme de simples nœuds de calcul. En d'autre termes, le gestionnaire de ressources ne tient pas compte des spécificités de la machine virtuelle qui est créée et intégrée par un administrateur.

Par conséquent, cette utilisation de machines virtuelles ne tire réellement pas partie de certaines propriétés de la virtualisation qui seraient notamment intéressantes dans notre contexte. En fait, grâce à la facilité de configuration des machines virtuelles ou leur propriété d'isolation, cela permettrait de :

- Simplifier l'allocation et la désallocation dynamiques de ressources.
- Faciliter la configuration dynamique des environnements d'exécution.
- Faciliter la prise en charge de l'exécution simultanée de plusieurs tâches sur un nœud, tout en assurant un niveau élevé de cloisonnement entre elles.

3.4 Synthèse : de la gestion de ressources mutualisées

De cet état de l'art, il ressort que la mise en place d'une plateforme de calcul doit s'accompagner de méthodes ou mécanismes adéquats pour bien exploiter la puissance de calcul délivrée. En effet, selon les objectifs à atteindre, les problématiques de gestion de ressources sont diverses avec différents niveaux de complexité.

Dans le contexte de mutualisation de ressources que nous nous sommes proposés d'étudier dans cette thèse, les objectifs d'exploitation sont difficiles à atteindre (gestion dynamique de tâches, partage proportionnel et utilisation efficace de ressources). Répondre aux problématiques que cela souleve nécessite notamment de disposer de mécanismes pour assurer, aussi bien l'allocation et la configuration dynamique de ressources, qu'un partage de ressources avec des contraintes très fortes sur l'équité et l'utilisation. Nous avons vu que les gestionnaires de ressources développés dans le contexte des grappes de calcul traditionnelles ne conviennent pas.

Pour répondre à ces problématiques, il est donc indispensable de développer de nouveaux outils. Dans cette optique, la virtualisation parait offrir des propriétés intéressantes pour traiter dynamiquement les tâches et aussi pour optimiser l'utilisation de ressources.

Toutefois, au regard de ce que nous avons étudiés jusqu'à présent, nous devons répondre à un certain nombre de questions :

- L'ordonnancement de machines virtuelles : Quel mécanisme de gestion de ressources devons-nous mettre en place pour satisfaire les règles de partage et d'utilisation de ressources imposées par la mutualisation ? Puisque nous avons vu que les gestionnaires de ressources existants ne conviennent pas.
- L'exécution dynamique de tâches par le biais d'une machine virtuelle : Comment gérer l'exécution dynamique d'une tâche au sein d'une machine virtuelle qui devra, de manière automatique, rapide et transparente, être créée, personnalisée et déployée pour l'exécution ? Précisons en effet que, d'une tâche à l'autre les configurations aussi bien logicielles (comme par exemple l'image système et les outils logiciels ou les données) que matérielles (CPU ou mémoire) varient.
- Gestion du cycle de vie des machines virtuelles sur la grappe : Comment gérer le cycle de vie des machines virtuelles (création, déploiement, démarrage, préemption, redémarrage, migration) au niveau de la grappe, de manière aussi transparente que possible ? Sinon il serait difficile d'assurer la dynamicité espérée de la plateforme.
- Performance des applications : Quelle architecture ou quels outils de virtualisation devons nous mettre en œuvre pour minimiser, voire réduire, les dégradations de performances induites par la virtualisation ?

C'est à ces questions que nous allons répondre au fil des prochains chapitres.

Mutualisation d'une plateforme pour des services de calcul à la demande 4

4.1 Présentation du problème

Nous rappelons que dans cette thèse nous visons à définir une solution de nuage de services de calcul intensif à la demande destinée à des petites et moyennes entreprises ou industries (PME/PMI). Nous l'avions vu, la mise en place d'un nuage de calcul nécessite une plateforme de calcul adéquate. Or, l'accès aux moyens de calcul intensif est un enjeu très souvent hors de portée pour ces entreprises. Cela étant essentiellement dû au fait que l'acquisition et l'exploitation d'une telle plateforme est onéreuse pour une PME/PMI qui dispose d'un budget de fonctionnement faible.

Pour une telle entreprise, l'utilisation de ressources avec paiement à la demande via un nuage constitue une alternative intéressante. Mais la location de ressources en mode IaaS par exemple reste chère en emploi sur le long terme. Par ailleurs, les plateformes PaaS du marché ciblent les applications web et présentent des limites pour supporter des services de calcul intensif. En outre, les nuages de calcul offrent peu de souplesse concernant la gestion et le contrôle des ressources. Le client n'a pas accès à certaines informations, liées par exemple à la localisation géographique des nœuds ou aux interconnexions réseaux, qui peuvent être utiles pour optimiser les performances des applications.

Visant à réduire l'impact budgétaire pour chacune des entreprises, la mutualisation constitue une solution intéressante. Elle pose cependant des problématiques de gestion de ressources complexes que nous devons traiter :

- Partager proportionnellement des ressources entre les services des différentes entreprises ;
- Maximiser l'utilisation de ces ressources ;
- Allouer et restituer dynamiquement des ressources pour assurer le traitement automatiquement des tâches ;
- Prendre en compte des priorités entre des tâches : pour un service donné, les tâches pourraient avoir des priorités différentes. Mais d'un service à l'autre, les priorités des tâches ne sont pas comparables.

4.2 Vers une plateforme mutualisée pour des services de calcul

Nous avons observé au chapitre précédent que la virtualisation offrait de bonnes propriétés pour répondre au problème posé. Et au regard des questions que cela soulève, nous avons besoin de :

- *Dynamicité* dans l'allocation de ressources avec pour optique d'assurer aisément le déploiement automatique des tâches.
- *Flexibilité* du mécanisme de gestion de ressources afin d'assurer un partage proportionnel des ressources de la grappe entre les différents services sans pour autant sacrifier l'utilisation.

Nous apporterons des solutions qui couvrent les points suivants :

Architecture virtualisée pour l'exécution dynamique de tâches : Nous définirons une architecture où les tâches sont exécutées au sein de machines virtuelles. Ce choix de la virtualisation permettra, d'une part, de fournir via les machines virtuelles un support très utile pour partitionner un nœud entre plusieurs tâches tout en garantissant un niveau de cloisonnement (ou d'isolation) élevé entre des tâches s'exécutant au sein des machines virtuelles distinctes. D'autre part, cette virtualisation servira de socle pour la gestion dynamique de tâches, les machines virtuelles lui apportant une souplesse et plusieurs autres propriétés qui permettent de créer et configurer automatiquement les environnements d'exécution des tâches.

Mécanisme de partage de ressources flexible : Nous proposerons également un mécanisme de gestion de ressources qui exploite aussi les propriétés de la virtualisation : en particulier, la politique d'affectation de ressources repose sur une notion de baux de ressources où l'exécution d'une tâche est réalisée via un bail géré par une machine virtuelle.

Les prochaines sections fourniront plus de détails sur ces différents aspects. Pour information, notons que notre démarche suppose que les applications sont séquentielles, multiprogrammées ou multithreads, c'est-à-dire des applications non distribuées.

4.3 Virtualiser pour favoriser la dynamicité et l'automatisation

L'exécution de chaque tâche se fera au sein d'une machine virtuelle qui constitue donc l'unité d'exécution. Pour cela, nous aurons besoin de manipuler des machines virtuelles sur les sur les nœuds de la grappe. Pour permettre la dynamicité et la flexibilité attendue de la plateforme, cela implique de pouvoir manipuler à la volée tout le cycle de vie d'une machine virtuelle (création, déploiement, démarrage, arrêt, destruction, suspension, redémarrage et migration).

Nous définirons une architecture qui, pour gérer le cycle des machines virtuelles au niveau de la grappe, reposera sur un outil de gestion d'infrastructure virtuelle existant. Ce dernier sera étendu avec une composante logicielle spécifique chargée de fournir les fonctionnalités pour exploiter la grappe conformément aux objectifs fixés.

Vu de manière abstraite, l'architecture logicielle (figure 4.1) intègre une composante spéciale appelée *gestionnaire SaaS*, qui fournit une abstraction SaaS au dessus d'un gestionnaire

d'infrastructure virtuelle qui est nativement orienté IaaS. Au sein de l'architecture, le gestionnaire d'infrastructure virtuelle est intégré comme une boite à outils pilotée par le gestionnaire SaaS. En effet, pour exécuter une tâche, ce dernier s'appuie sur le premier pour créer de manière dynamique et transparente la machine virtuelle associée. Au plus bas niveau, un hyperviseur déployé sur les nœuds sert de socle pour la virtualisation.

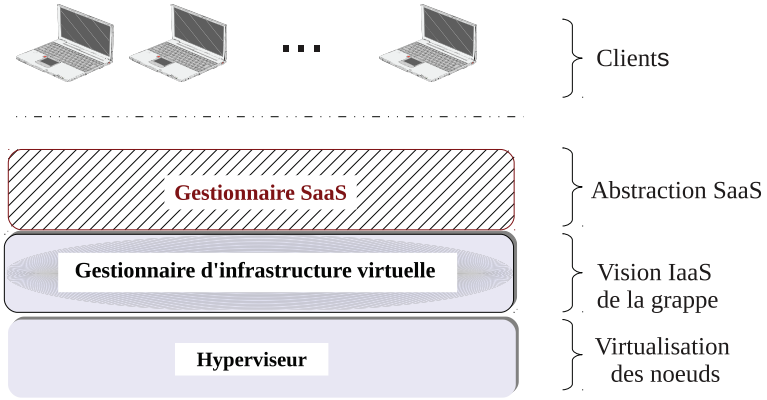

FIGURE 4.1 – Vision abstraite de l'architecture logicielle.

L'architecture est consituée de composante qui offre des mécanismes et des algorithmes qui permettent d'assurer :

- La réception ou l'admission des requêtes.
- L'ordonnancement de tâches, qui inclut entre autres le lancement, la préemption, la reprise et la terminaison. Ce qui implique la gestion du cycle de vie des machines virtuelles associées à leur exécution (création et personnalisation, placement, démarrage, suspension, redémarrage, migration et arrêt).
- La gestion des entrées-sorties : l'architecture intègre pour cela un mécanisme de stockage de données en réseau, qui permet de simplifier et d'accélérer la manipulation dynamique des machines virtuelles.
- L'observation ou le contrôle de la plateforme en cours de fonctionnement. Ce qui implique de superviser les tâches, les machines virtuelles et les nœuds au fil du temps.

Cette architecture impose toutefois des exigences liées notamment au gestionnaire d'infrastructure virtuelle et à l'hyperviseur :

Exigences sur la gestion d'infrastructure virtuelle : elle doit supporter des mécanismes de personnalisation automatique (contextualisation , voir page 41) pour simplifier la prise en charge du déploiement dynamique de l'environnement d'exécution des différentes tâches. En effet, pour chaque tâche, l'environnement d'exécution est différent à plusieurs égards : (i) la pile logicielle (image système et outils logiciels), (ii) les données à traiter, ainsi que (iii) la configuration matérielle (comme le nombre de CPU, la quantité mémoire et l'adressage réseau de la machine virtuelle).

Pour illustrer les spécificités à prendre en compte, si nous considérons trois services, *Service App*$_i$, $i = 1 \ldots 3$, comme à l'exemple de la figure 3.3. Alors, les piles logicielles nécessaires à l'exécution de leurs différentes seraient typiquement différentes. De plus, les données à traiter, ainsi que certains paramètres matériels comme le nombre de

CPU et la taille de la mémoire affectés à la machine virtuelle, varieraient d'une tâche à l'autre car l'application sous-jacente peut demander un ou plusieurs processeurs pour les traiter.

Toutes les machines virtuelles sont créées à partir d'une image de base. Pour créer la machine virtuelle nécessaire à l'exécution d'une tâche donnée, cette image est dynamiquement dupliquée et automatiquement personnalisée. Cette personnalisation est faite de telle sorte que la tâche soit lancée au démarrage de la machine virtuelle. Pour une tâche nécessitant n CPU/cœurs ($n \geq 1$), le système doit créer une machine virtuelle ayant n CPU virtuels et allouer n cœurs physiques pour son exécution. Cette machine virtuelle doit également être détruite après que la tâche soit terminée.

Exigences sur l'hyperviseur : Il doit être compatible avec le gestionnaire d'infrastructure virtuelle et être suffisamment performant pour minimiser les dégradations de performances induites par la virtualisation. C'est pourquoi nous nous appuierons sur un hyperviseur paravirtualisé (ex. Xen, Hyper-V ou VMmware ESXi) et nous orienterons vers une approche virtualisation hybride. En rappel, cela consiste à combiner la paravirtualisation et la virtualisation assistée par le matériel pour optimiser les performances des machines virtuelles.

Ainsi virtualisée, notre architecture présente plusieurs intérêts :

- Permettre l'exécution simultanée de plusieurs tâches sur un nœud tout en assurant un degré élevé d'isolation entre elles, car chaque tâche est exécutée au sein d'une machine virtuelle. Cela apporte également une flexibilité qui permet d'optimiser l'allocation et l'utilisation de ressources.
- De plus, la possibilité de suspendre et de redémarrer ultérieurement une machine virtuelle permet de gérer de manière efficace la préemption des tâches best-effort. En effet, étant plus simple et moins contraignant qu'un mécanisme de type *checkpointing*, la suspension et le redémarrage de machines virtuelles évite de perdre des calculs déjà réalisés par une tâche préemptée.
- Enfin, en nous appuyant sur des outils existants de virtualisation, nous évitons d'avoir à redévelopper des fonctionnalités de base de manipulation de machines virtuelles, ni au niveau de la grappe, ni au niveau des nœuds. Nous nous concentrons donc sur nos objectifs tout en bénéficiant des fonctionnalités et les performances que ces outils de virtualisation, qui ont été testés et éprouvés, offrent.

4.4 Allocation et gestion flexibles de baux de ressources virtuelles

Le mécanisme d'allocation de ressources repose sur deux notions fondamentales : la notion de types de tâches (tâche cliente ou tâche best-effort) et la notion de bail de machine virtuelle. Dans la suite, nous préciserons d'abord ces notions qui permettront de mieux appréhender le principe de l'allocation en soi.

4.4.1 Types de tâches : tâches de service et tâches best-effort

A partir des règles de mutualisation, nous distinguons deux classes de tâches avec en prime deux niveaux de priorités d'accès aux ressources :

Tâches de services ou tâches de production : une tâche de service correspond à l'exécution d'une requête de service soumise par un client. Nous considérons qu'une tâche de production ne peut être suspendue ou arrêtée jusqu'à la fin de son exécution. De plus, nous devons prendre en compte des priorités entre des tâches liées à un service donné.

Tâches best-effort : Elles correspondent à toutes les autres tâches. Elles sont dites *tâches best-effort* parce qu'elles peuvent et doivent même être suspendues sous certaines conditions. Notamment, une ou plusieurs tâches best-effort doivent être suspendues lorsqu'il n'y a pas suffisamment de ressources pour exécuter une tâche de production qui attend.

4.4.2 Baux de machines virtuelles : bail préemptif et bail non-préemptif

L'exécution d'une tâche associée à une requête est gérée selon un *mécanisme de bail* où une machine virtuelle est réservée pour l'exécution de la tâche. Toutefois, il est important de noter que, contrairement à un système de bail classique (comme avec Haizea par exemple, voir page 40) où un bail est réservé par un utilisateur, le bail est entièrement géré dynamiquement et automatiquement par notre système :

- Le début d'un bail est déterminé dynamiquement par le système via une politique d'ordonnancement de tâches, conçue pour satisfaire les règles de partage et d'utilisation de ressources fixées par la mutualisation.
- La création, le déploiement et le démarrage de la machine virtuelle, ainsi que le démarrage de la tâche au sein de celle-ci sont pris en charge automatiquement par le système
- La fin d'un bail ou la durée du bail est fonction de la durée d'exécution de la tâche : la machine virtuelle est détruite aussitôt que la tâche est terminée pour libérer les ressources.

Suivant la classification des tâches (tâches de production ou best-effort) et leurs règles respectives d'accès aux ressources, nous distinguons deux types de baux de ressources :

Bail non-préemptif. Comme son nom l'indique, un bail non-préemptif n'est ni suspendu, ni arrêté, jusqu'à ce que la tâche affectée à la machine virtuelle auquel il est associé ait terminé son exécution.

Un bail non-préemptif est associé à toute machine virtuelle chargée de l'exécution d'une tâche de production puisqu'une telle tâche ne doit être ni suspendue, ni arrêtée avant la fin de son exécution. Dès qu'une tâche de production est démarrée, elle s'exécute jusqu'à la fin – à moins qu'elle ne soit interrompue accidentellement par un événement externe au système. Son bail est automatiquement arrêté dès que l'exécution de la tâche est terminée, la machine virtuelle est alors automatiquement détruite pour libérer les ressources qui lui ont été allouées.

Bail préemptif. C'est un bail qui peut être suspendu ou arrêté sous certaines conditions.

Un bail préemptif est associé à une machine virtuelle chargée de l'exécution d'une tâche best-effort. Ainsi considérant les contraintes d'exécution d'une tâche best-effort comme ci-dessus mentionnées, un bail préemptif est automatiquement suspendu (suspension de la machine virtuelle) pour faire passer une tâche de production qui est en attente par manque de ressources suffisantes pour son exécution. Le bail est automatiquement rétabli (redémarrage de la machine virtuelle) dès que des ressources redeviennent disponibles.

Dans un volet commercial, ces baux pourront de plus être utiles pour la facturation des services. Notamment la facturation du traitement d'une requête de service donnée prendrait en compte la quantité de ressources et la durée du bail consommées.

4.4.3 Un mécanisme de partage de ressources flexible

Par rapport à un découpage statique ou une politique de type *fair-share*, le mécanisme de partage proposé [54, 55] permet de maximiser l'utilisation de ressources tout en évitant des situations de longue attente « injuste » pour des tâches d'un service qui utilise peu de ressources. En outre, il limite l'impact d'un comportement égoïste où certains services pourraient abusivement surutiliser les ressources au détriment des autres.

Dans la suite, nous supposerons être en mesure de connaitre ou d'estimer le temps d'exécution de chaque tâche. Ce qui est une hypothèse réaliste vu que nous sommes face à des applications bien maitrisées, par leurs éditeurs qui sont aussi les fournisseurs de services. Pour illustration, nous avons montré dans [52] un exemple où, à partir des connaissances d'une application, un modèle d'estimation par approximation est établi à partir des données en entrée et des échantillons de temps d'exécution.

Le mécanisme d'allocation de ressources se résume comme suit :

1. A un service S_{e^i} d'une entreprise donnée e^i ($i = 1 \ldots n$, n correspondant au nombre total d'entreprises investissant sur la plateforme), nous associons un ratio d'utilisation r_{e^i} proportionnel à l'investissement de cette entreprise.
2. Pour limiter des situations de longue attente pour des tâches d'un service qui utilise peu de ressources :
   - L'allocation de ressources pour une tâche qui prendrait beaucoup de temps se fait comme si le partage était assuré suivant un découpage strict. En d'autres termes, une tâche qui prendrait beaucoup de temps ne sera démarrée que si et seulement si, après son démarrage, l'utilisation de ressources par le service auquel elle est liée sera inférieure à r_{e^i}.
   - Par contre, l'exécution d'une tâche qui prendrait peu de temps peut outrepasser cette règle sous réserve de certaines conditions. Notamment un dépassement du ratio d'utilisation ne doit se produire que si et seulement si aucune tâche en attente lié à un service qui utilise peu de ressources ne peut être exécutée.

   Pour atteindre ce but, nous distinguons deux types de tâches, *tâches courtes* et *tâches longues*, selon leur durée d'exécution. Par définition, contrairement à une tâche courte, une tâche sera dite longue si le temps requis pour son exécution est supérieur à une

durée *seuil* donnée. Ce seuil, noté ,D_{seuil}, est un paramètre fourni au système. Nous en discuterons davantage à la section 4.5.

3. Dans la file des tâches d'un service donné, la sélection de la tâche à exécuter tient compte des priorités. Mais le respect des priorités n'est pas strict car, pour optimiser l'utilisation de ressources, la sélection permet également le *backfilling* (cf. page 28).

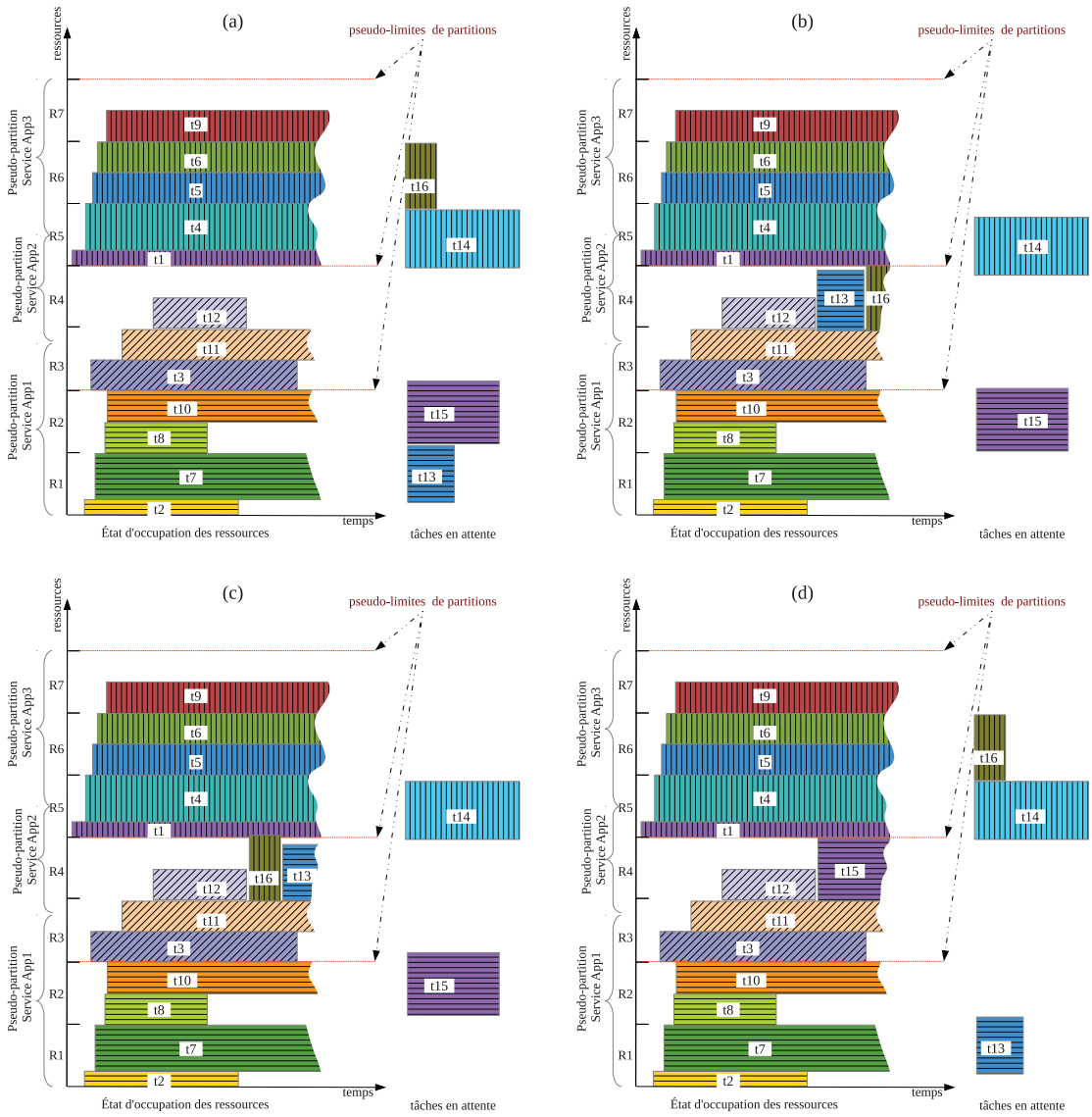

FIGURE 4.2 – Illustration du mécanisme d'allocation de ressources.

Illustration

Considérons l'exemple de la figure 4.2-*a* où, comme au chapitre précédent, nous avons imaginé une situation avec une plateforme composée de sept ressources, Ri, $i = 1 \ldots 7$, qui doivent être partagées pour des tâches de trois services, App_i, $i = 1 \ldots 3$. Cet exemple assume également les fournisseurs des différents services investissent respectivement à hauteur des

$2/7$, $2/7$ et $3/7$ sur la plateforme. Les tâches t_i, $i = 1 \ldots 16$, sont indiquées par des rectangles. Les tâches liées au service App_1 sont hachurées en traits horizontaux, celles liées au service App_2 en traits obliques, tandis que les tâches en traits verticaux sont liées au service App_3. La hauteur d'une tâche indique la quantité de cœurs demandée et sa longueur, sa durée.

Selon la valeur attribuée au paramètre D_{seuil}, imaginons que :

- Les tâches $t13$, $t14$, $t15$ et $t1$ (en attente) sont considérées comme de longues durées. Alors aucun changement ne sera réalité, et nous resterons dans la même configuration que sur figure 4.2-*a*.
- Les tâches $t13$ et $t16$ sont considérées comme de courtes durées : (i) Si la file du service App_1 a été sélectionnée en premier, alors la plateforme sera dans la configuration de la figure 4.2-*b*. La tâche $t13$ a été démarrée jusqu'à être terminée, tandis que la tâche $t16$ est encore en cours d'exécution. Nous voyons dans tous les deux cas que l'utilisation des ressources aura été accrue par rapport à un découpage statique. Et en même temps nous pouvons assurer que, à partir de chacune n'importe laquelle des configurations, si une tâche relative au service App_2 arrivait ou était en attente, elle serait prise en charge dans un délai inférieur ou égale à D_{seuil}. (ii) Si par contre c'est la file du service App_3 qui avait été sélectionnée en premier, nous aurons été conduit vers la situation de la figure 4.2-*c*. Et par similitude, le raisonnement précédent reste valable.
- Les tâches $t13$, $t14$, $t15$ et $t1-$ sont considérées comme de longues durées, mais que la tâche $t15$ est de type best-effort. Alors nous serons dans la configuration de la figure 4.2-*d* où la tâche $t15$ aura été démarrée. Cette tâche pourra être suspendue à tout moment pour exécuter une tâche de production d'un service qui remplit les critères du partage des ressources ainsi mutualisées.

4.5 Discussions et synthèse

Dans le meilleurs des cas, chaque entreprise devrait investir en fonction de ses besoins car son investissement détermine la quantité de ressources que ses tâches longues pourraient utiliser à la fois. Précisons en effet que la mutualisation vise à permettre une fédération de ressources dont la puissance de calcul résultante serait pleinement exploitée par complémentarité. D'où l'intérêt de ne pas importer des contraintes de partage fortes sur des tâches courtes et best-effort.

La complémentarité avec des tâches courtes est assurée grâce au paramètre D_{seuil} . Pour limiter l'impact de comportements égoïstes, ce paramètre doit être raisonnablement fixé par consensus entre les différents protagonistes de la mutualisation. Comme nous pouvons le constater, le choix de ce paramètre est essentiel :

- S'il est trop petit, la proportion de tâches courtes serait faible et nous reviendrons à un cas de partage strict.
- S'il est trop grand, cela expose le système à des comportements égoïstes.

Ceci conclut ce chapitre où nous avons présenté les idées directrices de notre démarche. Avant l'évaluation et la validation expérimentale qui seront faites au chapitre 7, nous décrirons au fil des deux prochains chapitres une mise en œuvre en nous s'appuyant sur OpenNebula comme gestionnaire d'infrastructure virtuelle, et Xen comme hyperviseur. Le chapitre

suivant fournira des détails concernant aussi bien l'architecture que les algorithmes de gestion de ressources. Le chapitre 6, quant à lui, présentera l'implémentation d'un prototype du gestionnaire SaaS.

Plateforme logicielle pour la mutualisation de ressources virtualisées pour des services de calcul 5

Dans ce chapitre nous allons d'abord, section 5.1, détailler l'architecture de la plateforme. La section 5.2 décrira le fonctionnement, de la réception d'une tâche à son exécution en passant par l'ordonnancement de tâches qui est assuré par notre mécanisme de gestion de ressources. Le chapitre se terminera par une conclusion qui va clore nos contributions avant leur évaluation au chapitre suivant.

5.1 Architecture

Cette section présente et discute des choix architecturaux aussi bien d'un point de vue logicielle que matérielle. Même si nous insistons davantage, comme dans le reste de la thèse, sur les aspects logiciels qui constituent l'essentiel de l'architecture.

5.1.1 Architecture logicielle

La figure 5.1 présente l'architecture logicielle de la plateforme. Elle est découpée en trois niveaux suivant les fonctions de ses principales composantes logicielles :

- Au niveau le plus élevé, le gestionnaire SaaS sert d'interface d'accès ou point d'entrée à la plateforme. C'est en outre la composante principale fournissant l'essentiel des fonctionnalités d'exploitation attendues telles que le partage de ressources, la planification de l'exécution des requêtes et l'allocation dynamique de machines virtuelles. Des détails aussi bien de son intégration que de son couplage avec OpenNebula seront décrites à la section 5.1.1.1.
- Nous retrouvons au niveau intermédiaire le gestionnaire d'infrastructure virtuelle OpenNebula : par rapport à ses concurrents, il semble avoir les caractéristiques appropriées par rapport à nos objectifs d'exploitation. En particulier, en plus de son mécanisme de contextualisation qui est d'un grand apport pour faciliter la création de machines virtuelles personnalisées, son architecture et ses APIs conçues nativement pour faciliter son intégration et sa personnalisation permet de créer et gérer automatiquement les machines virtuelles nécessaires aux exécutions des tâches. Plus de détails de son intégration dans l'architecture seront données à la section 5.1.1.2.

– L'hyperviseur Xen a été choisi pour gérer la virtualisation au plus bas niveau. Ce choix, au lieu d'un de ses concurrents tels que Hyper-V ou VMware ESXi/ESX – également supportés par OpenNebula –, a été motivé par diverses raisons : (i) par rapport à la technologie Hyper-V qui est encore naissante, la solution est suffisamment mature. (ii) Concernant VMware, il ne supporte pas toutes les fonctionnalités offertes par OpenNebula : en particulier, la fonctionnalité de contextualisation, qui est essentielle dans notre architecture, n'était pas supportée avec des machines virtuelles VMware [1] et en outre, (iii) Xen est une technologie open source et donc ouvert notamment pour des questions d'interopérabilité.

Ces différentes composantes sont détaillées dans les sous-sections suivantes.

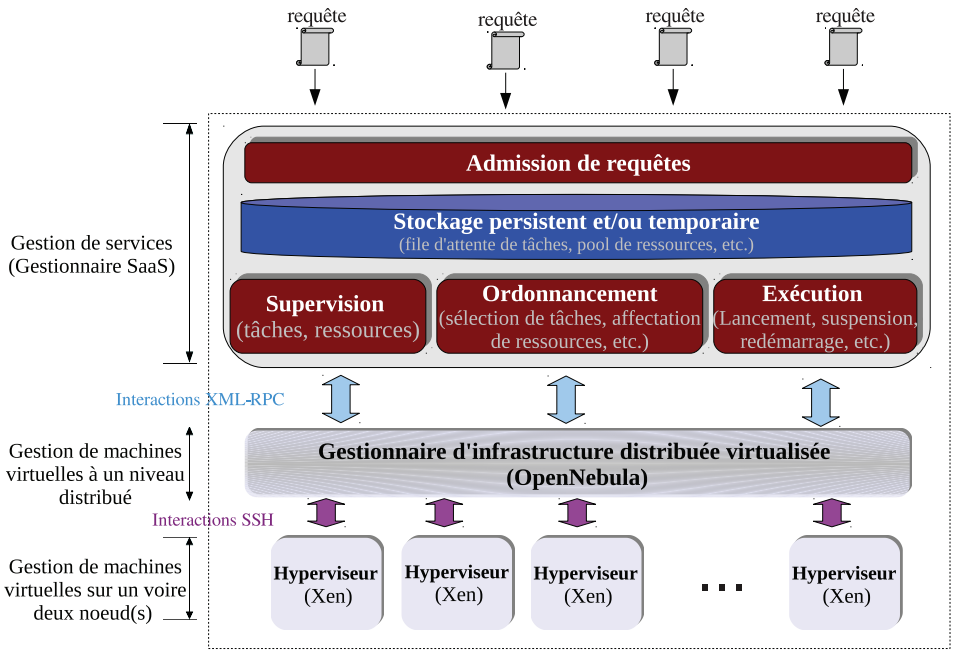

FIGURE 5.1 – Architecture logicielle.

5.1.1.1 Gestionnaire SaaS

La composante de gestion de services ou plus simplement le *gestionnaire SaaS* est un élément essentiel de notre architecture. Cette composante peut être vue comme un ensemble de quatre sous-systèmes logiques suivant ses fonctionnalités (figure 5.1). Ces fonctionnalités comprennent l'admission des requêtes, l'ordonnancement et l'exécution des tâches associées ainsi que la supervision ou le contrôle de l'état de la plateforme. En particulier, le sous-système d'ordonnancement de tâches implémente la stratégie de gestion de ressources conformément aux règles de partage et d'utilisation définies.

Pour exécuter une tâche, une machine virtuelle est créée automatiquement à partir d'une configuration générée automatiquement par le gestionnaire SaaS. Toutes les opérations de

1. Ce n'est que depuis la version 3.2 d'OpenNebula, qui n'a été publiée que pendant que ce manuscript était en fin de rédaction, que cette fonctionnalité est réellement opérationnelle : au début et durant la grande partie de nos travaux, la version 1.4.2 était la plus récente.

création, de déploiement ou de démarrage de la machine virtuelle sont initiées par le gestionnaire SaaS. Mais elles sont effectivement réalisées par OpenNebula.

Les interactions avec OpenNebula se font *exclusivement* via son API XML-RPC. Le gestionnaire SaaS implémente un client XML-RPC à cet effet. Remarquons que selon ce schéma d'intégration, le code du gestionnaire SaaS est séparé de celui d'OpenNebula. Cela implique donc un faible couplage avec OpenNebula contrairement à un cas où des modifications de code seraient nécessaires pour réaliser ce couplage. Ce qui aurait été le cas si nous avions, choisi un outil tel que OpenStack ou Eucalyptus.

Pour information, nous aurions également pu envisager le couplage via son API OCCI, étant donné que cette interface émerge comme un standard dans la communauté. Mais le support de cette API dans OpenNebula est encore peu mature.

5.1.1.2 Virtualisation de la grappe sur un socle OpenNebula

OpenNebula, rappelons-le, fournit des fonctionnalités pour manipuler les machines virtuelles sur l'ensemble des nœuds de la grappe. Dans l'architecture, il est spécifiquement intégré pour être utilisé comme une boîte qui fournit et expose des fonctionnalités de manipulation de machines virtuelles au gestionnaire SaaS. Ainsi, OpenNebula n'initie de manière *autonome* aucune opération de manipulation d'une machine virtuelle car toute opération doit être commandée par le gestionnaire SaaS via une requête XML-RPC. Mais en définitif, l'opération est effectivement réalisée par OpenNebula qui s'appuie sur le gestionnaire de machine virtuelle déployé sur le nœud cible (Xen, dans ce cas).

Pour que l'ensemble marche correctement, nous avons désactivé l'ordonnanceur par défaut d'OpenNebula nommé *mm_sched*. Car dans son fonctionnement par défaut, *mm_sched* vérifie périodiquement s'il y a des machines virtuelles non-démarrées pour les démarrer. Sous réserve que les ressources libres soient suffisantes pour l'exécution d'au moins une de ces machines virtuelles.

Pour effectivement gérer les machines virtuelles, OpenNebula s'appuie sur le Dom0 de Xen déployé sur chaque nœud. Xen est en effet intégré comme une boite à outils qui expose à distance via SSH ses fonctionnalités de manipulation de machines virtuelles à OpenNebula : par exemple, pour créer une machine virtuelle (DomU) sur un nœud donné, OpenNebula invoque la commande appropriée via SSH. Cette commande est alors à proprement dit traitée par le Dom0 du nœuds en question.

5.1.2 Architecture de déploiement

Pour concevoir l'architecture physique, nous avons essentiellement pris en compte la facilité de création et de déploiement dynamique des machines virtuelles sur les nœuds de la grappe. C'est-à-dire qu'elle doit faciliter l'automatisation de la création des images de machines virtuelles de manière aisée, la gestion des entrées/sorties et des machines virtuelles, sans pour autant sacrifier les performances.

La figure 5.2 montre l'architecture physique, elle se compose :

- d'un ou deux serveurs où seront déployés le gestionnaire SaaS et OpenNebula. Précisons toutefois qu'il serait intéressant de séparer le serveur d'accès où sera installé le

gestionnaire SaaS du serveur d'OpenNebula. En effet, certaines opérations telle que la création d'une image de machine virtuelle nécessitent des écritures disque importantes lors de la duplication de l'image de base. Ainsi si le serveur hébergeant OpenNebula est très sollicité, la dégradation de performances pourrait impacter significativement le gestionnaire SaaS.

– Des nœuds d'hébergement de machines virtuelles sur lesquels l'hyperviseur Xen sera déployé. Chaque nœud d'hébergement doit idéalement être d'interconnecté à deux réseaux de manière à avoir un des réseaux dédiés aux machines virtuelles et l'autre pour les tâches relatives à la gestion de ces machines virtuelles (comme les commandes les communication avec le serveur OpenNebula).
– Deux systèmes de fichiers réseaux : l'un pour stocker les images de machines virtuelles et l'autre pour les données applicatives. Chacun des systèmes de fichiers peut être basé sur un dispositif NAS (*Network Attached Storage*, ex. NFS) ou SAN (Storage Area Network). Le gestionnaire SaaS configure l'image de chaque machine virtuelle de manière à ce que le dépôt de données soit automatiquement monté dans le système de fichiers de cette dernière au démarrage.

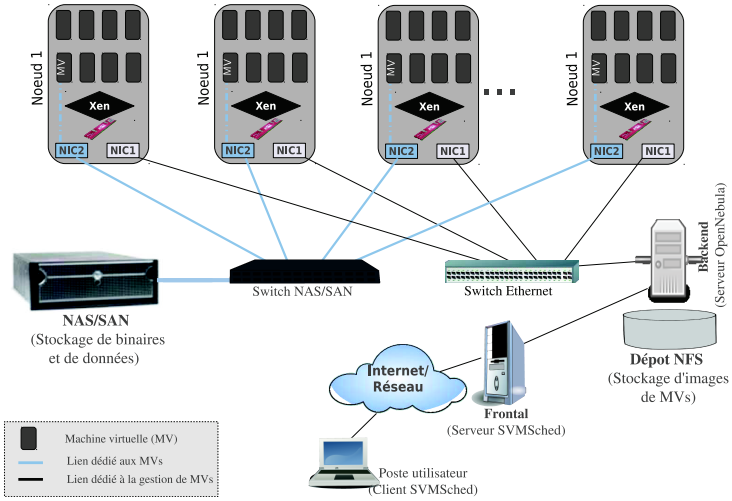

FIGURE 5.2 – Architecture physique.

Discussions

En étant basée sur un stockage de données en réseau, cette architecture facilite l'administration, la création et le déploiement rapides des machines virtuelles :

– Cela réduit le temps de création et de déploiement de machines virtuelles. Par exemple, au lieu d'empaqueter des données et des binaires dans l'image d'une machine virtuelle, nous pourrons plutôt définir des machines virtuelles de petites tailles qui pourront être automatiquement configurées au démarrage pour accéder au système de fichiers réseau où ces binaires et données auront été stockés.

– Si le dépôt repose sur un réseau haute performance tel que Gigabit Ethernet ou même Infiniband, cela améliorerait significativement les performances des lectures et des écritures de données en réseau.
– Enfin, grâce au stockage en réseau, nous pouvons nous assurer que si une machine virtuelle échouait accidentellement, les données des applications seront sauvées car étant stockées sur le système de fichiers réseau.

La centralisation de données n'a pas que des avantages, elle présente aussi des inconvénients : par exemple, des accès concurrents importants au système de fichiers centralisé entrainerait potentiellement une dégradation de performances. Aussi, une panne du système de fichiers entraînerait également une indisponibilité de toute la plateforme.

Néanmoins, ces problèmes peuvent être résolus, au moins partiellement, via des systèmes de fichiers parallèles par exemple. De même que des redondances des dispositifs de stockage contribuerait à renforcer leur robustesse.

5.2 Fonctionnement de la plateforme : Focus sur la gestion de ressources

Le gestionnaire SaaS gère quatre sous-systèmes gérant respectivement l'admission de requêtes, l'ordonnancement de tâches, l'exécution de tâches et enfin, la supervision de la plateforme (tâches, machines virtuelles et nœuds physiques).

Brièvement, avant que ces sous-systèmes ne soient détaillés dans la suite, mentionnons que l'admission intervient à l'arrivée d'une nouvelle requête et sert à analyser et valider la requête. Pour toute requête valide une tâche est créée et mise dans une queue d'attente pour être prise en charge plus tard par le sous-système d'ordonnancement, qui détermine quand est-ce qu'elle doit être exécutée. L'ordonnancement repose essentiellement sur des *algorithmes de sélection de tâches* qui déterminent, en tenant compte des règles de mutualisation, quelle tâche doit être démarrée à un instant donné. Chaque tâche sélectionnée est prise en charge par le sous-système d'exécution qui se charge de démarrer son exécution au sein d'une machine virtuelle qu'il aura crée, configuré et déployé à la volée sur un nœud de la plateforme. Enfin, le sous-système de supervision fournit des fonctionnalités de base pour contrôler de manière proactive l'état des ressources (nœuds et machines virtuelles) et des tâches qu'elles exécutent. En pratique, une supervision évoluée doit être réalisée par un outil externe adaptée comme Nagios, Ganglia, etc.

L'admission de requêtes et la gestion des tâches associées reposent sur un système à queues multiples où chaque service (ou entreprise) dispose de deux queues de tâches, une dédiée aux tâches de production et la seconde aux tâches best-effort (figure 5.3).

D'un point organisationnel, et toujours dans l'optique de simplifier des aspects de gestion de tâches, chaque service pourrait en outre disposer d'autres couples de structures de données – séparant toujours des tâches de production et best-effort – dans lesquelles seront stockées les tâches selon leur état d'exécution (en cours d'exécution, terminée, échouée, etc). Le nombre de couples de queues supplémentaires dépend des choix d'implémentation : nous pouvons par exemple avoir un seul couple de queues supplémentaire pour tous ces

autres états, ou plus simplement un couple de queues pour chaque état. Nous supposerons dans la suite que des tâches dans ces autres états sont stockées dans des structures distinctes des tâches *en attente*.

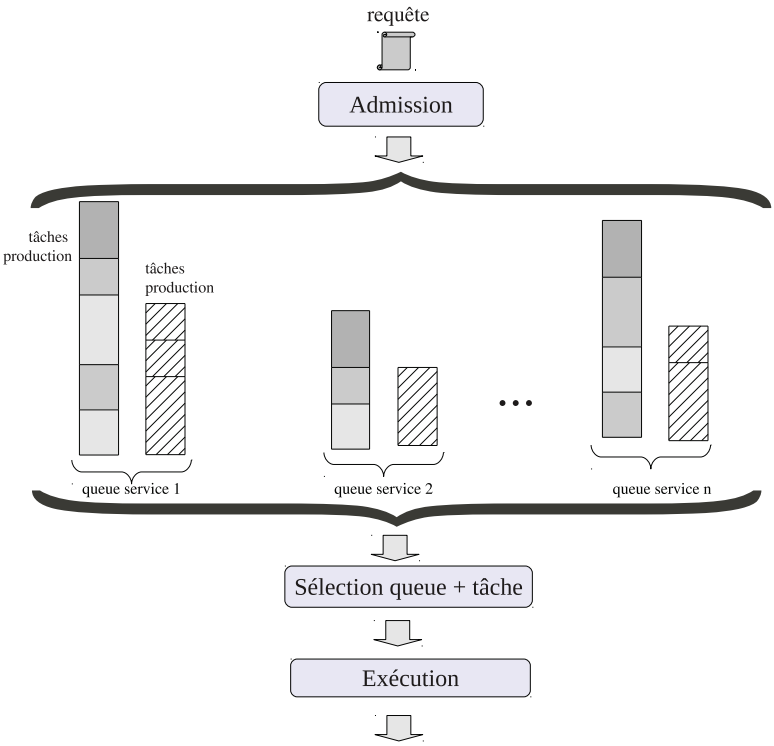

FIGURE 5.3 – Illustration du système de queues avec trois services. Chacun service dispose d'un couple de queues pour les tâches de production et best-effort (hachurée ou non).

5.2.1 Admission de requêtes

L'admission se résume comme suit :

1: Réception et analyse de la requête pour vérifier si elle est valide ou non.

2: Si elle est valide, créer la tâche associée, la marquer comme en *attente* (nous reviendrons sur le cycle de vie d'une tâche à la section 5.2.5) et la mettre en queue : pour déterminer la queue dans laquelle la tâche sera stockée, le système prend en compte le type de requête (production ou best-effort) et le service auquel cette requête est liée. En outre, pour simplifier les algorithmes d'ordonnancement comme nous le verrons à la section 5.2.2, les tâches sont ajoutées dans une queue de production de manière à garder la queue triée suivant l'ordre de priorité des tâches. Alors que les tâches best-effort sont stockées suivant leur ordre d'arrivée.

3: Sinon, la requête est rejetée et une notification renvoyée au client.

5.2.2 Sélection de tâches

Cette section décrit respectivement les algorithmes de sélection des tâches de service et des tâches best-effort.

5.2.2.1 Sélection de tâches de services

Suivant le système de queues multiples, la sélection d'une tâche de production qui considère l'ensemble des queues de production se déroule comme suit :

1. Sélection d'une queue éligible parmi l'ensemble des queues de tâches production. La sélection de queue se fait en round-robin : supposer que la queue de service de l'entreprise e^i notée q_{e^i} est élue à la date d, alors à la date $d+1$ c'est la queue $q_{e^{i+1}}$ de l'entreprise e^{i+1} qui sera élue.
2. Recherche dans la queue élue une tâche qui peut être exécutée tout en respectant les règles de partage de ressources établies pour la mutualisation.

Résumé par l'algorithme 1, la sélection sélection d'une tâche de service consiste à itérer sur les queues (boucle externe) puis sur les tâches (boucle interne), et au cours de chacune des itérations de la boucle interne :

- La fonction *prendreQueue()*, lignes 3 et 21, assure la sélection des queues en round-robin, conformément à la description précédente.
- La fonction *trouverNoeudPourExecuter()*, ligne 9, recherche dans l'ensemble des nœuds un nœud ayant suffisamment de ressources libres ou utilisée en best-effort pour supporter l'exécution de la tâche.
  Cette fonction implémente l'algorithme 2, qui commence d'abord par vérifier si la tâche peut être exécutée à partir des ressources inutilisées. Si, à l'issu de l'étape précédente aucun nœud n'est trouvé, il vérifie s'il y a des tâches best-effort en cours d'exécution dont la suspension pourrait permettre d'exécuter la tâche.
  La sélection de la tâche à suspendre est assez simple : l'idée est de suspendre la tâche la plus récente car des tâches qui s'exécutent depuis longtemps seraient presque en fin d'exécution. Remarquons dans la seconde boucle de l'algorithme 2 que la liste des nœuds est parcourue dans le sens inverse.
  En somme, si nous considérons que le nombre total des nœuds vaut N alors la complexité de cette fonction est de l'ordre de $O(N)$.
- La fonction *calculcerUsageEscompte()*, ligne 10, calcule la proportion de ressources utilisées par un service.
  L'équation 5.1 illustre le principe du calcul. Les w_i sont des coefficients assignés à chaque type de ressources de calcul. Même si cette relation ne considère que des ressources processeurs et mémoires, elle pourrait très bien s'étendre aux autres types de ressources comme la bande passante réseau : en réalité, inspirée du modèle de calcul des facteurs de priorité entre les tâches dans le gestionnaire de ressources Maui, cette relation peut être vue comme si les autres facteurs w_i étaient nuls.

$$usage = w_1 * mem\_usage + w_2 * cpu\_usage \tag{5.1}$$

Notons, enfin, que les tâches best-effort ne sont pas comptabilisées dans ce calcul. Etant donné qu'elles ne sont exécutées que sur des ressources inutilisées et sont suspendues aussitôt qu'une tâche de production manque de ressources pour s'exécuter.

Algorithme 1 Algorithme de sélection d'une tâche de production

PRÉCONDITIONS: Queues de tâches triées par ordre de leur priorité : soient j et j' deux tâches, q une queue de tâches production, et Q l'ensemble de toutes les queues de tâches de production tel que $j \neq j' \in q \in Q$, $priorite(j)$ une fonction qui retourne la priorité d'une tâche donnée, $dateSoumission(j)$ qui retourne la date de soumission d'une tâche donnée et $rang(j,\ q)$ qui retourne le rang d'une tâche j dans une queue q. Si $priorite(j) < priorite(j')$ ou $priorite(j) = priorite(j')$ tant dis que $dateSoumission(j) < dateSoumission(j')$ Alors $rang(j,\ q) < rang(j',\ q)$.

POSTCONDITIONS: L'algorithme se termine avec l'une des post-conditons suivantes : La variable *trouve* vaut *vrai* si une tâche est sélectionnée ; la variable *tacheSelectionnee* contient alors la description de la tâche sélectionnée. La valeur de la variable *reglesPartageRespectees* (qui peut être *vrai* ou *faux*) permet de savoir si les contraintes de partage ont été outrepassées par une tâche courte ou pas. Une valeur *faux* de la variable *trouve* signifie que toutes les queues ont été parcourues sans succès. Alors, on a l'une des conditions suivantes qui est satisfaite : (i) toutes les queues sont vides ; (ii) aucune tâche ne peut être exécutée tout en étant conforme aux contraintes de partage imposées par l'algorithme d'ordonnancement.

```
1:  trouve ← FAUX
2:  reglesPartageRespectees ← FAUX
3:  queue ←prendreQueue(Q_p)
4:  TANT QUE ( NON reglesPartageRespectees ET queue != finEnsembleQueue(Q_p)) FAIRE
5:     service ← serviceAssocie (queue)
6:     usageAut ← usageAutorise (service)
7:     tache ← teteQueue (queue)
8:     TANT QUE ( NON reglesPartageRespectees ET tache != finQueue (queue) ) FAIRE
9:        SI ( trouverNoeudPourExecuter(tache) ) ALORS
10:          usageEsc ← calculcerUsageEscompte (tache, service)
11:          SI (usageEsc ≤ usageAut) OU
                (usageEsc > usageAut ET estUneTacheCourte(tache)) ALORS
12:             tacheSelectionnee ← tache
13:             trouve ← VRAI ;
14:             SI ( usageEsc ≤ usageAut ) ALORS
15:                reglesPartageRespectees ← VRAI
16:             FIN SI
17:          FIN SI
18:       FIN SI
19:       tache ← prendreTache ( queue )
20:    FIN TANT QUE
21:    queue ← prendreQueue (Q_p)
22: FIN TANT QUE
```

D'un point de vue complexité globale, nous pouvons résumer l'algorithme 1 comme suit : si m représente le nombre moyen de tâches dans les différentes queues, K le nombre de queues (services) et N le nombre de nœud de la plateforme. Alors, la complexité de cet algorithme est de l'ordre de $O(K * m * N) = O(m * N)$, sachant qu'en pratique K sera

constant et idéalement très petit $\ll 10$ – pour simplifier les contraintes de coopération pour la mutualisation.

Algorithme 2 Sélection du nœud d'hébergement d'une tâche de production

PRÉCONDITIONS: L'algorithme prend en entrée, une tâche $tache$ et l'ensemble des nœuds de calcul $listeNoeud$.

POSTCONDITIONS: Si la variable *trouve* vaut *vrai* alors un nœud ayant suffisamment de ressources matérielles a été trouvé. La variable *noeud* contient alors la description du nœud sélectionné. Sinon, on a atteint la fin du pool de nœuds.

```
1:  trouve ← FAUX
2:  preemptionRequired ← FAUX
3:  noeud ← teteListeNoeud(listeNoeud)
4:  TANT QUE ( trouve = FAUX ) FAIRE
5:     SI (memoireLibre(noeud) ≥ memoireRequise(tache) ET
                 cpuLibre(noeud) ≥ cpuRequis (tache) ) ALORS
6:        trouve ← VRAI ;
7:     SINON
8:        noeud ← prochainNoeud( listeNoeud )
9:     FIN SI
10: FIN TANT QUE
11: SI NON trouve ALORS
12:    noeud ← finListeNoeud(listeNoeud)
13:    TANT QUE ( trouve = FAUX ) FAIRE
14:       memoireUtile ← memoireLibre(noeud) + memoireBestEffort(noeud)
15:       cpuUtile ← cpuLibre(noeud) + cpuBestEffort(noeud)
16:       SI ( memoireUtile ≥ memoireRequise(tache) ET
                 cpuUtile ≥ cpuRequis(tache) ) ALORS
17:          trouve ← VRAI ;
18:       SINON
19:          noeud ← prochainNoeudHost(listeNoeud)
20:          preemptionRequired ← VRAI
21:       FIN SI
22:    FIN TANT QUE
23: FIN SI
```

5.2.2.2 Sélection de tâches best-effort

Si aucune tâche ne peut être sélectionnée à l'issu de l'algorithme 2, le système cherchera à exécuter une tâche best-effort (s'il y en a en attente). Le principe est simple par rapport à la sélection d'une tâche de production. En effet, selon les règles de la mutualisation, cela implique très peu de contraintes à respecter (ex. une tâche best-effort n'est exécutée que sur des ressources inutilisées et l'allocation de ressources pour une telle tâche n'est pas sujette au respect des règles de partage).

Le pseudo code de l'algorithme 3 utilisé pour cela se déroule en deux étapes, comme lors de la sélection d'une tâche de production : (i) la sélection d'une queue parmi les différentes queues de tâches best-effort, et ensuite, (ii) la sélection d'une tâche au sein de cette queue. La

sélection de queue se fait en round-robin, comme dans le cas des tâches de production. Mais la sélection de la tâche à exécuter au sein de cette queue repose sur un algorithme *premier arrivé, premier servi* (FCFS) avec remplissage (*backfilling*).

Algorithme 3 Sélection d'une tâche best-effort

PRÉCONDITIONS: Queues de tâches triées suivant leur date d'arrivée ou de soumission : soient j et j' deux tâches, q une queue de tâches best-effort, et Q l'ensemble de toutes les queues de tâches best-effort tel que $j \neq j' \in q \in Q$, $dateSoumission(j)$ une fonction qui retourne la date de soumission d'une tâche donnée, $rang(j,\ q)$ une fonction qui retourne le rang d'une tâche j dans une queue q. Si $dateSoumission(j) < dateSoumission(j')$ Alors $rang(j,\ q) < rang(j',\ q)$.

POSTCONDITIONS: Si la variable *trouve* vaut *vrai* alors la variable *tacheSelectionnee* contient la description la tâche sélectionnée. Sinon, cela signifie qu'aucune tâche n'a été sélectionnée pour l'une des raisons suivantes : (i) toutes les queues sont vides ; (ii) il n'y a pas de ressources libres ; (ii) aucune tâche best-effort en attente ne peut être exécutée avec les ressources actuellement libres.

1: $trouve \leftarrow$ FAUX
2: $queue \leftarrow$ prendreQueue (Q)
3: TANT QUE ($reglesPartageRespectees$ = FAUX) ET ($queue$!= finQueue_set(Q)) FAIRE
4: $service \leftarrow$ recupererIdService ($queue$)
5: $tache \leftarrow$ prendreTacheEnTete ($queue$)
6: TANT QUE ($tache$!= finQueue ($queue$)) ET ($reglesPartageRespectees$ = FAUX)) FAIRE
7: SI (trouverNoeudPourExecuter($tache$)) ALORS
8: $queueSelectionnee \leftarrow queue$
9: $tacheSelectionnee \leftarrow tache$
10: $trouve \leftarrow$ VRAI
11: FIN SI
12: $tache \leftarrow$ prendreProchaineTache($queue$)
13: FIN TANT QUE
14: $queue \leftarrow$ prendreQueue (Q)
15: FIN TANT QUE

5.2.3 Exécution de tâche : Création transparente de machine virtuelle

Pour exécuter une tâche le système crée, de manière réactive et transparente vis-à-vis du client, une machine virtuelle dans laquelle il démarre l'exécution. Le processus de démarrage d'une tâche se résume comme suit :

1: Préparation de la configuration de la machine virtuelle : un modèle de configuration, appelé *template*, dans le jargon OpenNebula, est automatiquement générée. S'il faut le rappeler, ce *template* comprend des paramètres dynamiques spécifiques à chaque tâche comme par exemple le nombre CPU virtuel, la taille de la mémoire, l'adressage réseau (IP et MAC) et les paramètres de lancement et d'exécution de la tâche (comme les données d'entrée, l'application à exécuter (chemin d'accès du binaire), l'accès au système de fichiers réseau et la commande de lancement de la tâche,).

2: Instanciation ou création proprement dite de la machine virtuelle dans la volée. En effet, après la 1, la machine virtuelle est telle que la tâche se lance automatiquement au démarrage de la première.

3: Déploiement/démarrage de la machine virtuelle sur un nœud sélectionné au préalable pour cet effet.
4: Vérification du démarrage de la machine virtuelle et de la tâche. Après le démarrage, la tâche est marquée comme *en exécution* et déplacée dans la queue appropriée.

5.2.4 Contrôle des exécutions et supervision de la plateforme

La supervision est une fonctionnalité nécessaire dans la majorité des plateformes informatiques. Elle sert à connaitre de manière proactive ou passive l'état des services (applications) et des ressources. Dans notre cas, les informations sont collectées de manière proactive pour connaitre l'état des tâches, des machines virtuelles et des nœuds physiques :

- Nœuds : les informations collectées sont notamment la disponibilité (actif ou non-actif), les quantités de cœurs et mémoires disponibles, le taux d'utilisation des processeurs et de la mémoire et le nombre de machines virtuelles actives.
- Machines virtuelles : les informations collectées sont la disponibilité, les quantités de cœurs virtuels et mémoire allouées, le nœud d'hébergement, le taux d'utilisation des processeurs et de la mémoire et le nombre de machines virtuelles actives.
- Tâches : les informations récupérées sont le statut (active, terminée, échouée...), la machine virtuelle d'hébergement, les dates de soumission et de démarrage, la classe et le type de tâche (tâche de production ou tâche best-effort, tâche courte ou tâche longue).

En particulier, la supervision génère des statistiques sur l'utilisation de ressources, aussi bien de manière globale que par les différents services. De telles statistiques peuvent, par exemple, être utilisées sur un long ou moyen terme pour la gestion de la capacité des ressources de la plateforme (observation des comportements passés et présents pour faire des prévisions d'évolution future).

5.2.5 Cycle de vie d'une tâche

Suivant le modèle présenté ci-dessus, chaque requête soumise sur la plateforme a un cycle de vie illustrée sur la figure 5.4.

Les états *Suspension*, *Suspendu*, *Redémarrage*, et *Migration* sont spécifiques aux tâches best-effort car les opérations conduisant à ces états ne sont applicables qu'à ces tâches.

Les transitions d'états fonctionnent comme suit :

- L'état *Attente* est l'état initial de toute tâche.
- De l'état *Attente*, la tâche passe à l'état *Démarrage* après qu'elle ait été sélectionnée par la composante d'ordonnancement et prise en charge par celle d'exécution. Elle reste dans ce dernier état jusqu'à ce que son démarrage soit effectif.
- A la fin du démarrage, elle passe à l'état *Exécution*.
- Si pour une raison quelconque le démarrage échoue, elle passe à l'état *Echec* qui est un état final ou bloquant : un état à partir duquel aucune transition n'est possible.
- Depuis l'état *Exécution*, une tâche peut passer à l'un des états *Terminé*, *Inconnu*, *Suspension* ou *Migration*, selon des conditions qui décrites par les quatre points suivants.

- Le changement de l'état *Exécution* vers l'état *Terminé* (état final) se produit lorsque la tâche a correctement terminé son exécution.
- La transition de l'état *Exécution* vers l'état *Inconnu* (état final) se produit lorsque, pour une raison quelconque (ex. lorsque la machine virtuelle associée n'est plus accessible), le système n'est plus en mesure de récupérer les informations sur la tâche.
- La transition de l'état *Exécution* vers l'état *Suspension* intervient juste avant lorsqu'une tâche doit être préemptée (suspendue).
- Une tâche passe de l'état *Exécution* à l'état *Migration* dans les conditions suivantes : (i) elle a été redémarrée (après une suspension) et ne peut disposer d'un nombre de cœurs ou une quantité de mémoire suffisants, par rapport à ce qu'elle demande, sur le nœud initial (où elle a été suspendue) ; (ii) un second nœud ayant suffisamment de cœurs et de mémoire pour l'accueillir est disponible. En précisant que toutes ces conditions sont détectées par le système de manière transparente, la tâche est migrée sur ce second noeud : si la migration – qui est automatiquement orchestrée immédiatement après que la machine virtuelle ait démarrée – se déroule correctement, la tâche repasse à l'état *Exécution*. Sinon (échec de la migration), elle passe à l'état *Echec*.
- La transition de l'état *Suspension* à l'état *Suspendu* se passe si la suspension de la tâche se termine correctement. Sinon, elle passerait à l'état *Echec*. Un tel échec peut avoir diverses raisons comme par exemple une perte connexion réseau ou une interruption brusque de la machine virtuelle.
- Depuis l'état *Suspendu* la seule transition possible est celle vers l'état *Redémarrage*, vers laquelle transite une tâche best-effort suspendue qui doit être redémarrée. Si le redémarrage se déroule correctement, la tâche passe à l'état *Exécution*, sinon elle passe à l'état *Echec*.

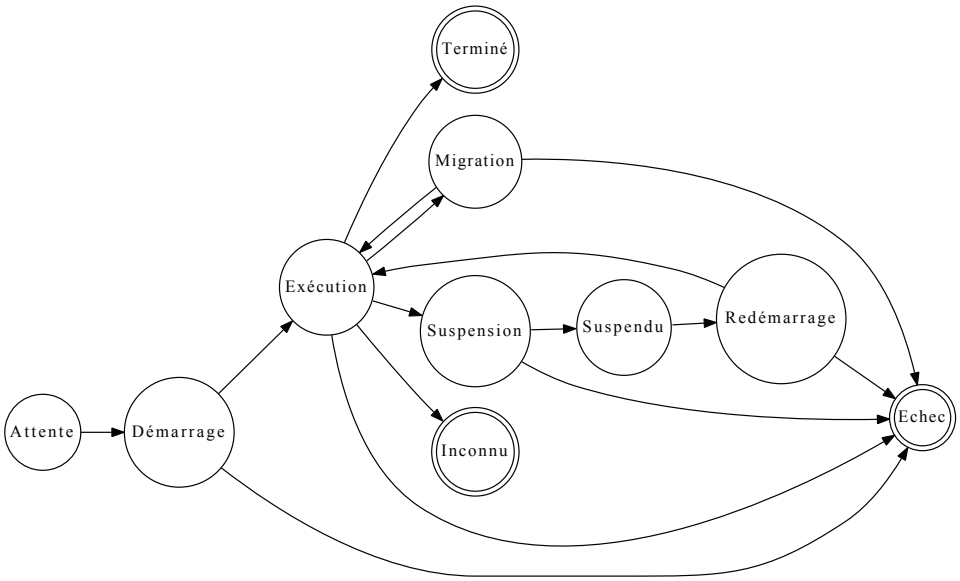

FIGURE 5.4 – Cycle de vie d'une tâche

5.3 Synthèse

Dans ce chapitre, nous avons d'abord discuté de l'architecture et présenté le fonctionnement d'une plateforme de mutualisation de ressources pour de service de calcul. Dans le modèle considéré les tâches associées aux requêtes de service sont exécutées au sein de machines virtuelles. L'architecture s'appuie et exploite des fonctionnalités d'OpenNebula et Xen, deux outils de virtualisation éprouvés via un gestionnaire de ressources spécifique au sein duquel sont mises en œuvre des fonctionnalités nécessaires à la mutualisation.

Le gestionnaire SaaS doit assurer parmi d'autres fonctionnalités l'admission des requêtes (réception, validation, création de tâches...), l'ordonnancement et l'exécution des tâches associées, la supervision de composants et ressources de la plateforme (tâches, machines virtuelles, nœuds physiques...).

Dans le chapitre suivant, nous décrirons le prototype du gestionnaire SaaS développé dans l'objectif de valider nos contributions.

SVMSched : un prototype du gestionnaire de services 6

Ce chapitre présente SVMSched [1] (*Smart Virtual Machine Scheduler*) [55, 54, 51], un prototype du gestionnaire SaaS que nous avons développé en utilisant des langages et bibliothèques C/C++. C'est un système composés de plusieurs modules logiciels plus ou moins autonomes. Outre les différentes fonctionnalités, concernant notamment la gestion des règles de partage et d'utilisation de ressources dans un contexte de mutualisation, l'ensemble est conçu avec à l'esprit une facilité d'intégration et de prise en main.

Dans la section suivante, nous décrirons d'abord son architecture interne avant de présenter, section 6.2, quelques détails importants concernant son implémentation. Nous terminerons le chapitre en présentant nos conclusions.

6.1 Composantes architecturales

L'architecture de SVMSched est illustrée par la figure 6.1 qui fait ressortir ses principales composantes et leurs interactions. Comme nous pouvons remarquer, cette architecture repose sur plusieurs processus qui sont mis en évidence sur la figure par des ellipses. Assurant chacun un rôle spécifique, ces processus peuvent être permanents ou transitoires : d'une part, un processus permanent ou démon (ellipse en trait dense) a une durée de vie égale à celle du système. Et d'autre part, un processus transitoire ou *agent* a quant à lui une durée de vie limitée dans le temps : cela correspond au temps requis par un traitement spécifique et ponctuel comme, par exemple, l'analyse et la validation d'une requête.

6.1.1 Point d'accès

C'est la composante chargée de la réception, la validation et l'enregistrement des nouvelles requêtes provenant des clients.

A l'arrivée d'une requête (1), un processus appelé *Contrôleur d'accès* la réceptionne et crée un agent *Validateur* qui est un processus temporaire à qui il délègue la charge de l'analyse et de la validation de la requête (2) – L'analyse et la validation des requêtes simultanée se fait en parallèle comme illustré sur la figure 6.1. Les résultats de cette étape de validation sont sauvegardés (3) de manière persistante dans une base de données tout en gardant des traces dans un *journal d'événements* dont un extrait est présenté à la figure 6.2.

1. `https://gforge.inria.fr/projects/svmsched`

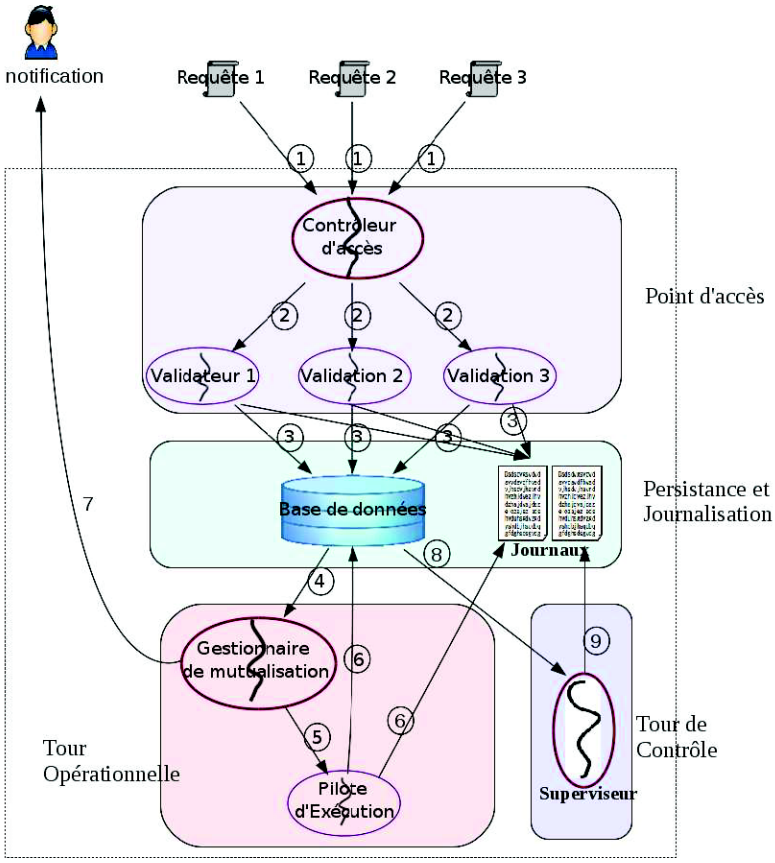

FIGURE 6.1 – Architecture et flot de traitement de SVMSched.

Chaque évènement journalisé dans ce fichier est indicé par le temps, en seconde, écoulé depuis le démarrage du système. Sur l'extrait de la figure 6.2, par exemple, les lignes mises en évidence par l'encadré indiquent des informations concernant :

- La réception puis la validation d'une requête (ex. à la date 195).
- La sélection d'une tâche puis l'assignation à une machine virtuelle créée à la volée (lignes étiquetées par *[SCHED][I]* à la date 195).
- Le démarrage de la machine virtuelle sur le nœud sélectionné à cet effet (lignes étiquetées par *[SCHED][I]* aux dates 195 et 210).

6.1.2 Gestionnaire de mutualisation

Intégré au sein de la composante *Tour Opérationnelle* qui assure la planification et l'exécution des tâches, le gestionnaire de mutualisation est un démon spécial chargé de la sélection d'une tâche conformément aux règles de mutualisation.

Après avoir sélectionné une tâche (4), le gestionnaire de la mutualisation crée un processus temporaire appelé *Pilote d'exécution*, auquel il délègue la charge de préparer et de démarrer l'exécution de la tâche dans une machine virtuelle (5). Après le démarrage il met à jour la base de données basée et sauvegarde des traces dans le journal des événements (6). Lorsque la tâche est terminée, le gestionnaire de mutualisation qui est le point central de cette composante opérationnelle notifie le client (7).

```
0 Fri Oct  8 06:45:46 2010 [CORE][I]: Listener successfully started
0 Fri Oct  8 06:45:46 2010 [CORE][I]: Scheduler successfully started
0 Fri Oct  8 06:45:46 2010 [CORE][I]: Monitor successfully started
0 Fri Oct  8 06:45:46 2010 [SCHED][I]: No production job to schedule, trying to find best-effort jobs
0 Fri Oct  8 06:45:46 2010 [SCHED][I]: Neither production nor best-effort job to schedule
195 Fri Oct  8 06:49:01 2010 [RQHANDLER][I]: Parsing new request: svmschedclient -r jivarod -a 1145 -n 4 -t prod -T 1145
195 Fri Oct  8 06:49:01 2010 [RQHANDLER][I]: New job has been added in queue, Job ID: 1
195 Fri Oct  8 06:49:01 2010 [SCHED][I]: Execute the job 1. Job type: prodution. Command: svmschedclient -r jivarod -a 1145 -n 4 -t prod -T 1145
195 Fri Oct  8 06:49:01 2010 [SCHED][I]: The job 1 will be executed within the virtual machine 0
195 Fri Oct  8 06:49:01 2010 [SCHED][I]: Starting the virtual machine 0 on edel-13.grenoble.grid5000.fr
210 Fri Oct  8 06:49:16 2010 [SCHED][I]: Virtual machine 0 started
210 Fri Oct  8 06:49:16 2010 [SCHED][I]: No production job to schedule, trying to find best-effort jobs
210 Fri Oct  8 06:49:16 2010 [SCHED][I]: Neither production nor best-effort job to schedule
212 Fri Oct  8 06:49:18 2010 [RQHANDLER][I]: Parsing new request: svmschedclient -r jivarod -a 40 -n 1 -t prod -T 40
212 Fri Oct  8 06:49:18 2010 [RQHANDLER][I]: New job has been added in queue, Job ID: 2
212 Fri Oct  8 06:49:18 2010 [SCHED][I]: Execute the job 2. Job type: prodution. Command: svmschedclient -r jivarod -a 40 -n 1 -t prod -T 40
212 Fri Oct  8 06:49:18 2010 [SCHED][I]: The job 2 will be executed within the virtual machine 1
212 Fri Oct  8 06:49:18 2010 [SCHED][I]: Starting the virtual machine 1 on edel-13.grenoble.grid5000.fr
227 Fri Oct  8 06:49:33 2010 [SCHED][I]: Virtual machine 1 started
227 Fri Oct  8 06:49:33 2010 [SCHED][I]: No production job to schedule, trying to find best-effort jobs
227 Fri Oct  8 06:49:33 2010 [SCHED][I]: Neither production nor best-effort job to schedule
246 Fri Oct  8 06:49:52 2010 [RQHANDLER][I]: Parsing new request: svmschedclient -r jivarod -a 67 -n 1 -t prod -T 67
246 Fri Oct  8 06:49:52 2010 [RQHANDLER][I]: New job has been added in queue, Job ID: 3
246 Fri Oct  8 06:49:52 2010 [SCHED][I]: Execute the job 3. Job type: prodution. Command: svmschedclient -r jivarod -a 67 -n 1 -t prod -T 67
246 Fri Oct  8 06:49:52 2010 [SCHED][I]: The job 3 will be executed within the virtual machine 2
246 Fri Oct  8 06:49:52 2010 [SCHED][I]: Starting the virtual machine 2 on edel-13.grenoble.grid5000.fr
262 Fri Oct  8 06:50:08 2010 [SCHED][I]: Virtual machine 2 started
262 Fri Oct  8 06:50:08 2010 [SCHED][I]: No production job to schedule, trying to find best-effort jobs
262 Fri Oct  8 06:50:08 2010 [SCHED][I]: Neither production nor best-effort job to schedule
266 Fri Oct  8 06:50:12 2010 [RQHANDLER][I]: Parsing new request: /mnt/context/svmschedclient -c 1/2
266 Fri Oct  8 06:50:12 2010 [SCHED][I]: The action 'shutdown' will be performed on the virtual machine 1
272 Fri Oct  8 06:50:18 2010 [RQHANDLER][I]: Parsing new request: svmschedclient -r bof -a 64 -n 4 -t prod -T 64
272 Fri Oct  8 06:50:18 2010 [RQHANDLER][I]: New job has been added in queue, Job ID: 4
272 Fri Oct  8 06:50:18 2010 [SCHED][I]: Execute the job 4. Job type: prodution. Command: svmschedclient -r bof -a 64 -n 4 -t prod -T 64
272 Fri Oct  8 06:50:18 2010 [SCHED][I]: The job 4 will be executed within the virtual machine 3
272 Fri Oct  8 06:50:18 2010 [SCHED][I]: Starting the virtual machine 3 on edel-14.grenoble.grid5000.fr
287 Fri Oct  8 06:50:33 2010 [SCHED][I]: Job completed, Job ID: 2
287 Fri Oct  8 06:50:33 2010 [SCHED][I]: Virtual machine 3 started
287 Fri Oct  8 06:50:33 2010 [SCHED][I]: No production job to schedule, trying to find best-effort jobs
287 Fri Oct  8 06:50:33 2010 [SCHED][I]: Neither production nor best-effort job to schedule
326 Fri Oct  8 06:51:12 2010 [RQHANDLER][I]: Parsing new request: svmschedclient -r jivarod -a 108 -n 1 -t prod -T 108
326 Fri Oct  8 06:51:12 2010 [RQHANDLER][I]: New job has been added in queue, Job ID: 5
326 Fri Oct  8 06:51:12 2010 [SCHED][I]: Execute the job 5. Job type: prodution. Command: svmschedclient -r jivarod -a 108 -n 1 -t prod -T 108
326 Fri Oct  8 06:51:12 2010 [SCHED][I]: The job 5 will be executed within the virtual machine 4
326 Fri Oct  8 06:51:12 2010 [SCHED][I]: Starting the virtual machine 4 on edel-13.grenoble.grid5000.fr
328 Fri Oct  8 06:51:14 2010 [RQHANDLER][I]: Parsing new request: /mnt/context/svmschedclient -c 2/3
328 Fri Oct  8 06:51:14 2010 [SCHED][I]: The action 'shutdown' will be performed on the virtual machine 2
341 Fri Oct  8 06:51:27 2010 [SCHED][I]: Virtual machine 4 started
341 Fri Oct  8 06:51:27 2010 [SCHED][I]: No production job to schedule, trying to find best-effort jobs
341 Fri Oct  8 06:51:27 2010 [SCHED][I]: Neither production nor best-effort job to schedule
346 Fri Oct  8 06:51:32 2010 [SCHED][I]: Job completed, Job ID: 3
```

FIGURE 6.2 – Extrait d'un journal d'événements de SVMSched.

Ce qui n'est pas apparent sur ce schéma est le fait que chaque machine virtuelle embarque de manière transparente un binaire qui permet de notifier le gestionnaire de mutualisation lorsque la tâche est terminée. Ce dernier crée alors un processus temporaire qui se charge de détruire proprement la machine virtuelle et de libérer les ressources qu'elle occupait. Toutes ces opérations sont, bien entendu, suivi d'une mise à jour de la base de données et du journal des événements.

Par ailleurs, même si la phase (5) est réalisée par un processus indépendant, elle ne se fait pas en parallèle : avant de sélectionner une autre tâche, le gestionnaire de mutualisation attend que le pilote d'exécution termine son opération. D'une part, cela permet de garder un état cohérent sur l'utilisation de ressources en ayant la certitude que la tâche (et donc la machine virtuelle associée) a été bien démarrée ou non. D'autre part, nous évitons de surcharger le serveur OpenNebula, notamment le système de fichiers où sont stockées les images de machines virtuelles. Ceci est dû au fait que le nombre de duplications simultanées de l'image de base des machines virtuelles doit être contrôlé pour garantir des performances constantes.

6.1.3 Tour de contrôle

La composante *Tour de contrôle* (figure 6.2.) est essentiellement composée d'un démon *Superviseur* qui collecte et analyse périodiquement des données relatives aux tâches, aux machines virtuelles et aux nœuds physiques. A partir de cette analyse, il produit des métriques notamment liées à l'usage de ressources par les différents services. Les résultats sont enregistrés dans un *journal de supervision* dont un extrait est présenté à la figure 6.3. L'administrateur a ainsi la possibilité de surveiller l'utilisation de la plateforme.

Sur l'extrait de figure 6.3, par exemple, nous avons mis en évidence des métriques élémentaires concernant l'utilisation de ressources par des tâches relatives à un service nommé *jivarod*. La légende *total/prod/beff* (dans la partie surlignée) indique des métriques sur l'ensemble des tâches et les différentes classes de tâches (production, best-effort), respectivement.

```
1201 Fri Oct  8 01:20:39 2010 [MONITOR][I]:
*******************************   BEGIN REPORT   ***********************************************
Usable host(s): 8
CPU: 64
MEM: 196529 MB

HOSTNAME                          CPU      MEMORY    CPU USED    MEMORY USED    RUNNING VMs       STATE
-----------------------------------------------------------------------------------------------------------
edel-47.grenoble.grid5000.fr        8       24566           8            512              2        runn
edel-46.grenoble.grid5000.fr        8       24566           8            512              2        runn
edel-45.grenoble.grid5000.fr        8       24566           8            512              2        runn
edel-12.grenoble.grid5000.fr        8       24566           8            512              2        runn
edel-11.grenoble.grid5000.fr        8       24566           8            512              2        runn
edel-26.grenoble.grid5000.fr        8       24566           8            256              1        runn
edel-10.grenoble.grid5000.fr        8       24566           0              0              0        runn
adonis-9.grenoble.grid5000.fr       8       24567           0              0              0        runn

Unusable Host(s): 0
CPU: 0
MEM: 0 MB

Job(s) related to the service jivarod (total/prod/beff): 11/11/0
            jivarod CPU usage: 40/40.00/0.00         Ratio (%): 62.50/62.50/0.00
            jivarod Memory usage (MB): 2560/2560/0         Ratio (%): 0.00/1.30/0.00
            jivarod Granted usage (%): 67.71

JOB       DATE                        SERVICE                TYPE     CPU  MEMORY   STATE   VM ID  VM STATE    INPUT
-----------------------------------------------------------------------------------------------------------------------
1         Fri Oct  8 01:01:29 2010    jivarod            prodution    4.00     256    runn       0      runn    1145
7         Fri Oct  8 01:06:23 2010    jivarod            prodution    4.00     256    runn       6      runn    1662
10        Fri Oct  8 01:08:15 2010    jivarod            prodution    4.00     256    runn       9      runn    1088
11        Fri Oct  8 01:09:49 2010    jivarod            prodution    4.00     256    runn      10      runn    1057
12        Fri Oct  8 01:12:10 2010    jivarod            prodution    4.00     256    runn      11      runn    1174
14        Fri Oct  8 01:15:00 2010    jivarod            prodution    4.00     256    runn      13      runn    1146
15        Fri Oct  8 01:15:25 2010    jivarod            prodution    4.00     256    runn      14      runn    3428
16        Fri Oct  8 01:16:00 2010    jivarod            prodution    4.00     256    runn      15      runn    1901
17        Fri Oct  8 01:16:35 2010    jivarod            prodution    4.00     256    runn      16      runn    912
18        Fri Oct  8 01:16:49 2010    jivarod            prodution    4.00     256    runn      17      runn    890
20        Fri Oct  8 01:20:30 2010    jivarod            prodution    4.00     256    pend      -1      unkn    1248
2         Fri Oct  8 01:01:46 2010    jivarod            beff         1.00     256    pend       1      unkn    45448
5         Fri Oct  8 01:03:40 2010    jivarod            prodution    1.00     256    done       4      unkn    108
6         Fri Oct  8 01:05:55 2010    jivarod            prodution    1.00     256    done       5      unkn    54
8         Fri Oct  8 01:07:59 2010    jivarod            prodution    4.00     256    done       7      unkn    521

Job(s) related to the service bof (total/prod/beff): 1/1/0
                bof CPU usage: 8/8.00/0.00         Ratio (%): 12.50/12.50/0.00
                bof Memory usage (MB): 256/256/0         Ratio (%): 0.00/0.13/0.00
                bof Granted usage (%): 19.79

[FICHIER TRONQUE]
```

FIGURE 6.3 – Extrait d'un journal de supervision de SVMSched.

Il est à préciser que la base de données de SVMSched et celle d'OpenNebula sont complémentaires en termes d'informations qui y sont stockées ou collectées. En particulier, OpenNebula fournit un certain nombre d'informations notamment relatives :

– Aux machines virtuelles : nœuds de déploiement ou d'exécution, les quantités de res-

sources CPU et mémoire allouées, l'usage réel des ressources, etc.
- Aux nœuds physiques : noms d'hôtes, les quantités de ressources CPU et mémoires disponibles, allouées, ou réellement utilisées, le nombre de machines virtuelles actives, l'usage réel des ressources CPU et mémoire, etc.

Ces informations sont collectées via des interactions XML-RPC entre SVMSched et le démon d'OpenNebula qui se charge de les extraire de sa base de données [2].

A ces informations, SVMSched ajoute toutes les informations relatives à chaque tâche (quantités de CPU ou mémoire requises, machine virtuelle d'hébergement – si tâche démarrée, identifiant, commande d'exécution et les paramètres d'entrée) sont stockées dans celle de SVMSched.

6.2 Détails d'implémentation

SVMSched se compose de deux modules :
- Un module serveur (*svmsched*), dont l'architecture a été présentée ci-dessus. Son fonctionnement repose sur une fichier de configuration, présenté plus loin, qu'il charge au démarrage. Pour fonctionner en réseau, il offre grâce à sa composante *Point d'accès* une interface d'accès basée sur une socket TCP/IP. La persistance est réalisée via une base de données relationnelle SQLite [3], tandis que la journalisation, comme nous l'aurions à partir des extraits, est réalisée dans des fichiers ASCII.
- Un module client (*svmschedclient*), qui permet de soumettre des requêtes au serveur via le réseau. Dans un contexte de nuage où un service doit être accessible par Internet – typiquement à partir d'un navigateur – ce module pourra être interfacé par un formulaire web. A partir duquel les paramètres des requêtes seront positionnés.

Pour faciliter leur utilisation, chacun ces deux modules fournit une aide accessible en ligne de commande (option *-h*).

Dans la suite, nous décrirons la configuration du serveur et expliquerons comment les informations de configuration sont exploitées par le système, notamment pour gérer l'allocation dynamique d'une machine virtuelle pour l'exécution d'une tâche.

6.2.1 Configuration de SVMSched et définition des règles de mutualisation

SVMSched repose sur un fichier de configuration XML [93] *unique*, dont la syntaxe du fichier est décrite par un document de validation (DTD, *Document Type Définition*) spécifique. Définie par l'administrateur du système, une configuration appropriée est – comme l'illustre l'exemple de la figure 6.4 – structurée en trois sections (*Cloud*, et *AppServices*) servant respectivement à définir : des informations liées à l'infrastructure, la configuration par défaut d'une machine virtuelle, et les services.

2. Remarquons qu'SVMSched ne gère aucune spécificité liée à cette base de données qui peut donc évoluer de manière indépendante, aussi longtemps que les APIs de manipulation restent inchangées.
3. `http://www.sqlite.org/`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Config SYSTEM "svmsched.dtd" >

<Config>

<!-- Set connection to OpenNebula and other specific parameters -->
  <Cloud>
    <RpcUrl>http://opennebula@server:2633/RPC2</RpcUrl>
    <OneAuth>oneadmin:7bc8559a8fe509e680562b85c337f170956fcb06</OneAuth>
    <ShortJobThreshold>600</ShortJobThreshold>
    <RootDir>/cloud</RootDir>
    <AppPrefix>/cloud</AppPrefix>
  </Cloud>

<!-- Set VM Configuration/Contextualization Parameters-->
  <SvmDescription>
    <ServerAddress>svmsched@server</ServerAddress>
    <DefaultXenVmTemplate> /cloud/svmsched/etc/xenvm.tpl </DefaultXenVmTemplate>
    <ContextFiles>
      <ContextFile> /cloud/svmsched/etc/init.sh </ContextFile>
      <ContextFile> /cloud/svmsched/etc/svmschedclient </ContextFile>
    </ContextFiles>
  </SvmDescription>

<!-- Set Services and their configurations -->
  <AppServices>
<!-- Service 1 -->
    <AppService id="example-service1">
      <Software executable="/cloud/service1.bin" userid="uid1" type="sequential">
            Sofware-Service-1
      </Software>
      <DataServer>nfs@fileserver</DataServer>
      <DataRepository>/cloud</DataRepository>
      <Description>  This service belong to the Business 1 </Description>
      <ResourceAllocationPolicy weight="1">  <!-- => ratio_service_i =  weight_i / …
        <DefaultVmMemory>256</DefaultVmMemory>
        <DefaultVmCpu>1</DefaultVmCpu>
        <ParallelismLevel>1</ParallelismLevel>
      </ResourceAllocationPolicy>
    </AppService>
<!-- Service 2 -->
    <AppService id="example-service2">
      <Software executable="/cloud/service2.bin" userid="uid2" type="multithreaded"…
            Sofware-Service-2
      </Software>
      <DataServer>opennebula@server</DataServer>
      <DataRepository>/cloud</DataRepository>
      <Description> This service belong to the Business 2 </Description>
      <ResourceAllocationPolicy weight="1">  <!-- => ratio_service_i =  weight_i / …
        <DefaultVmMemory>256</DefaultVmMemory>
        <DefaultVmCpu>1</DefaultVmCpu>
        <ParallelismLevel>1</ParallelismLevel>
      </ResourceAllocationPolicy>
    </AppService>
  </AppServices>
</Config>
```

FIGURE 6.4 – Exemple de configuration de SVMSched.

Sur cette configuration, nous pouvons remarquer les élements suivants :

Définition des informations d'accès au serveur OpenNebula : les paramètres *RpcUrl* et *OneAuth* (de la section *Cloud*) indiquent respectivement l'adresse de connexion au serveur XML-RPC et une chaine de caractère contenant un nom (login) et un mot de passe (chiffré en MD5 [107]) pour s'authentifier sur le serveur.

Définition du paramètre D_{seuil} : il est global et définit à l'aide du paramètre *ShortJobThreshold* (en seconde) dans la section *Cloud*. Cependant, il est également possible à l'aide d'une option du module client (l'option *-s*) de forcer le système à considérer une tâche comme de courte durée.

Définition d'un Service : chaque service est défini dans un bloc *AppService* de la section *AppServices* : toute requête faisant référence à un service qui n'est pas défini dans cette section sera considérée comme invalide et donc rejetée par le système.

Chaque service a un ensemble de paramètres : remarquons, par exemple, la définition l'application sous-jacente (*Software*) avec ses d'attributs (chemin du binaire, un utilisateur système ayant suffisamment de droits d'exécution et d'accès aux données, le type d'application – qui peut-être séquentiel ou parallèle), l'adresse d'accès au serveur de fichiers NFS (*DataServer*).

Définition des règles de partage de ressources : elles sont établies pour chaque service dans la sous-section *ResourceAllocationPolicy*. En effet, l'attribut *weight* spécifie le poids à utiliser pour calculer le ratio maximal d'utilisation de ressources par des tâches longues du service concerné : pour un service S_{e_i} (d'une entreprise e_i donnée), ce ratio noté $RatioMax_{S_{e_i}}$ est calculé à partir de la relation :

$$RatioMax_{S_{e_i}} = \frac{weight_{S_{e_i}}}{\sum_{j=1}^{n} weight_{S_{e_j}}} \tag{6.1}$$

Où $weight_{S_{e_i}}$ est le poids affecté au service et $\sum_{j=1}^{n} weight_{S_{e_j}}$ la somme des poids affectés à l'ensemble des services.

6.2.2 Création et déploiement transparents d'une machine virtuelle

Pour supporter l'allocation et le déploiement dynamique de machines virtuelles, SVMSched fournit un client qui abstrait la configuration d'une machine virtuelle. Cette abstraction masque les spécificités de configuration d'une machine virtuelle et facilite par conséquent l'expression d'une requête. Par exemple, si nous considérons le fichier de configuration de la figure 6.4, une requête s'exprimerait sous la forme :

```
$ svmschedclient -H svmsched@server --vcpu=2 --memory=512 \
--run example-service2 --arg customer1-data.dat
```

Une telle requête se traduit par « *demander un service* example-service2 *pour le traitement d'un jeu de données* customer1-data.dat *avec une machine virtuelle ayant* 2 *CPU virtuels (*2 *cœurs physiques lui seront alloués) et* 512*Mo de mémoire alloués* » :

- *svmsched@server* désigne l'adresse IP ou le nom d'hôte de la machine où s'exécute le module serveur.
- Le type de la tâche (production ou best-effort), optionnel, peut être précisé par l'option `--type` ou `-t` en version courte : `--type=prod` pour une tâche de production (option par défaut) et `--type=beff` pour une tâche best-effort.
- Tel qu'indiqué précédemment, l'option `-s` permet de forcer le système à considérer une tâche comme de courte durée ; si rien n'est pas précisé, la tâche est considérée comme de longue durée.
- Après l'admission de la requête, SVMSched se chargera – au moment d'exécuter – la tâche, de créer, personnaliser et déployer à la volée une machine virtuelle adéquate pour son exécution.

6.3 Synthèse

Dans ce chapitre, nous avons présenté SVMSched, un prototype du gestionnaire SaaS proposé pour répondre au problème de mutualisation d'une plateforme de calcul, entre plusieurs fournisseurs de services de calcul à la demande. Nous avons notamment décrit son architecture, son fonctionnement et bien d'autres détails de son implémentation.

Dans le chapitre suivant, nous présenterons la méthodologie et les résultats des expérimentations conduites pour valider nos démarches et contributions.

Expérimentation et évaluation 7

Ce chapitre présente les résultats des expérimentations conduites pour évaluer notre solution sous deux aspects :

- Le premier aspect concerne les performances de l'architecture virtualisée qui seront évaluées en répondant aux questions telles que : quel est le cout/temps de création et de démarrage d'une machine virtuelle ? Est-il raisonnable ou trop grand ? Quelles sont les performances d'une application qui, s'exécutant au sein d'une machine virtuelle, accède aux données à partir d'un système de fichiers réseaux (NFS notamment) sur notre architecture ? Quelle dégradation cela induit-il ? Si dégradation, de quel facteur ? Ce facteur est-il acceptable ou non ?
- Le second aspect porte sur le mécanisme de partage de ressources sera évalué en répondant notamment à la question : respecte-t-il les règles de la mutualisation ? En d'autres termes, satisfait-il un partage proportionnel de ressources ? Favorise-t-il une meilleure utilisation de ces ressources ?

La prochaine section décrira l'environnement et les outils expérimentaux. Dans la section 7.2, nous décrirons les expériences et les résultats. Le chapitre se terminera par une discussion pour valider l'ensemble des résultats.

7.1 Conditions expérimentales

Cette section décrira d'abord les outils logiciels, ensuite l'environnement matériel et enfin les charges de travail (*Workload*) utilisées pour simuler des tâches.

7.1.1 Environnement logiciel et applications

Les expériences ont été réalisées à partir de :

OpenNebula 1.4.2 **:** Version la plus récente disponible pendant nos développements, SVM-Sched est développé à partir des spécifications de cette version [1].

Xen 3.4.2 **:** Aussi bien l'hyperviseur (*Dom0* dans le jargon Xen) que les machines virtuelles (*DomU*) reposaient sur des systèmes Linux Debian *lenny*. Paravirtualisé, les systèmes des machines virtuelles sont crées à partir d'une image de base de 500 mégaoctets (Mo) générée par le biais de l'outil *xen-create* de la suite *xen-tools* [2].

1. Mais il devrait normalement fonctionner avec les versions apparues ultérieurement puisque les APIs avec lesquelles les deux outils interagissent n'ont pas particulièrement changées
2. `http://xen-tools.org/`

Parsec [39], c'est une suite d'applications OpenMP spécialement conçues pour étudier les spécificités des architectures multicœurs. Comme benchmark, trois applications de la suite ont été utilisées afin d'étudier les performances des applications sur notre architecture :

- Bodytrack, une application de *tracking* d'image, dont les performances dépendent fortement des dispositifs d'entrée-sortie disque. En effet, son fonctionnement se résume à lire, traiter, et récrire des fichiers d'images.
- Freqmine, qui est une application de recherche de motifs dans un fichier ASCII. Ne nécessitant que deux d'opérations sur des fichiers (une lecture au début et une écriture à la fin du traitement), le plus gros de son traitement se fait en mémoire. L'application utilise notamment des structures de données dynamiques (de type map) dont la peut croitre ou décroitre au cours de l'exécution.
- Enfin, Blackscholes est une application de simulation de gestion d'options sur un marché financier. En termes d'exigences (lecture/écriture de fichier et opérations en mémoire), elle se situe entre les deux premières applications.

7.1.2 Environnement matériel et déploiement

Les expériences ont été réalisées à partir des nœuds du cluster *genepi* de la plateforme Grid'5000. Assez récent (datant de 2008), chaque nœud comprend deux CPUs Quad Xeon (2.27 Ghz) – huit coeurs, 8 gigaoctets de RAM, deux interfaces réseaux (Gibabit Ethernet 1 Gbit/s et Infiniband 20 Gbit/s) et un disque dur SATA d'une capacité de 160 gigaoctets.

En fonction des expériences (voir section suivante), nous avons réalité des déploiements impliquant jusqu'à 100 nœuds de calcul. Chaque déploiement est automatisé par le biais de scripts et d'environnements (images systèmes dans le jargon Grid'5000) spécifiques : à partir de l'un des sites de Grid'5000 (Grenoble dans le cas de nos expériences), ces scripts prennent en charge aussi bien la réservation des nœuds, leur déploiement et leur configuration conformément à l'architecture physique décrite à la page 69. Le déploiement de chaque composants de cette architecture (serveur OpenNebula, serveur SVMSched ou nœud d'hébergement de machine virtuelle) est réalisé à partir d'un environnement spécialement conçu à cet effet.

Cependant, ne disposant pas de dispositifs spéciaux de stockage comme indiqué par l'architecture, nous notons deux spécificités : les images de machines virtuelles sont stockées dans un système de fichiers NFS activé sur le nœud hébergeant OpenNebula, tandis que les données et les binaires d'applications sont stockées sur un système de fichiers NFS activé sur le nœud où tourne SVMSched.

Notons que ces outils de déploiement simplifient les expériences, mais surtout ils assurent en grande partie leur reproductibilité. Ils ont fait l'objet de tests ainsi que de démonstrations publiques, notamment lors de l'édition 2011 du Challenge Grid'5000 où nous avions d'ailleurs remporté le $1er$ Prix[3] avec une démonstration impliquant près de 100 nœuds sur le site de Nancy.

3. `https://www.grid5000.fr/mediawiki/index.php/Grid5000:News`

7.1.3 Charge de travail

Pour évaluer les algorithmes d'ordonnancement de tâches et de gestionnaire de ressources mis en place, nous avons eu besoin de simuler des charges de travail réalistes et de caractéristiques connues.

Nous avons pour cela employé des charges de travail synthétiques issues du fichier journal de la grappe de calcul SHARCNET [4].

En effet, comme dans nos exemples précédents nous avons défini trois services App_1, App_2 et App_3 et avons construit deux charges de travail d'échelle plus ou moins importante :

Petite échelle : Dans une phase préliminaire dont des résultats ont été publiés dans [55] et [54], nous avons construit une charge de travail de 96 tâches ayant les caractéristiques suivantes :
- 36% des 96 tâches ont une durée d'exécution inférieure à $30\,minutes$, la somme de la durée d'exécution de ces tâches représente un peu moins de 20% de la somme des durées des différentes tâches.
- 68%, respectivement 20% et 12% des tâches ont été associées au service App_1, respectivement au service App_2 et au service App_3.

Pour ces expériences, $D_{seuil} = 30\,minutes$, a été choisi comme seuil en dessous duquel une tâche est considérée comme étant de courte durée. La plateforme de test était constituée de 6 nœuds (soit l'équivalent de $6*8 = 48$ cœurs) dédiés à l'hébergement de machines virtuelles et 2 nœuds supplémentaires pour déployer OpenNebula et SVMSched.

Plus grande échelle : Pour étudier des problématiques de passage à échelle, nous avons cette fois-ci constitué une seconde charge de travail plus importante composée de 1000 tâches :
- 50%, respectivement 30% et 20% des tâches ont été associées au service App_1, respectivement au service App_2 et au service App_3.
- la durée maximale d'une tâche est inférieure à 2 heures ;
- 15% des tâches ont une durée d'exécution inférieure ou égale à $15\,minutes$, qui correspond à la valeur du paramètre D_{seuil} dans ce cas.

Dans ce cas, la plateforme de test était composée de 20 nœuds (équivalent à $20 * 8 = 160$ cœurs) dédiés à l'hébergement de machines virtuelles, tandis que OpenNebula et SVMSched ont été déployés sur deux autres nœuds indépendants.

Scénarios de soumission de tâches

Aussi à partir de scripts appropriés, les tâches sont soumises de deux façons à partir du *frontend* du site (frontend.grenoble.grid5000.fr) qui joue alors le rôle d'un poste client :
- La première façon consiste à maintenir, relativement à la charge de travail originale, les dates de soumission des différentes tâches. Cette façon de procéder permet d'observer le comportement du prototype comparativement à la charge de travail originale. Cependant, elle exige beaucoup de temps pour chaque expérience.

4. `http://www.cs.huji.ac.il/labs/parallel/workload/l_sharcnet/index.html`

– La deuxième façon consiste à soumettre toutes les tâches simultanément : en plus de réduire la durée de chaque expérience, cette dernière permet de voir en outre comment le système réagit face à une forte charge. Rappelons en effet que la prise en charge de chaque requête est réalisée par un processus système (agent *Validateur*) qui est crée dynamiquement à l'arrivée de la tâche.

7.2 Expériences et analyse des résultats

La section traitera dans l'ordre : l'évaluation des performances de l'architecture virtualisée, l'analyse du mécanisme de gestion de ressources et le passage à l'échelle.

7.2.1 Performances de la virtualisation

Cette évaluation de performances concernera le temps de démarrage d'une tâche et les surcouts de la virtualisation (observés sur des applications Parsec).

S'agissant des expériences, nous avons déployé un système de fichier NFS sur un nœud distinct, qui est monté dans chaque machine virtuelle pour supporter toutes opérations entrées-sorties. Notamment, les applications Parsec (Bodytrack, Blackscholes et Freqmine) ainsi que leurs données ont été installées sur ce système de fichiers.

7.2.1.1 Temps de démarrage d'une tâche

En rappel, le démarrage d'une tâche implique la création, la personnalisation et le déploiement d'une machine virtuelle à partir d'une image de machine virtuelle de 500 Mo.

Pour évaluer ce temps, nous avons d'abord mesurés à partir des fichiers journaux de SVMSched les temps de démarrage des tâches au cours des différentes expériences. Ensuite, la moyenne a été comparée avec le temps requis pour créer et déployer une machine virtuelle dont les données des applications sont empaquetés dans l'image : différents volumes de données allant de 1 Ko à 7.5 Go ont été considérés [5]. Dans ce second cas, ce temps correspond à une moyenne sur dix mesures.

Nous avons observé qu'un temps *constant* de 15.0 $secondes$ est nécessaire au gestionnaire SaaS pour démarrer une tâche (duplication de l'image, personnalisation et déploiement de la machine virtuelle) :

– A titre indicatif, ces 15.0 secondes ne représentent que 5% de la durée d'une tâche de 3 minutes et moins de 2% de celle d'une tâche de 15 minutes. Cela nous semble acceptable, voire négligeable par rapport à la durée d'exécution de beaucoup d'applications dans le domaine de calcul intensif.
– En comparaison avec le temps de création et de déploiement d'une machine virtuelle dont les données sont empaquetées dans l'image, nous observons sur la figure 7.1 que ces 15.0 secondes restent également très faibles. Pour 7.5 Go de données par exemple,

5. Cette intervalle a été arbitrairement choisie, mais reste réaliste car les volumes de données traités par beaucoup d'applications sont de plus en plus importants atteignant plusieurs gigaoctets. C'est par exemple la cas de l'application jivarod souvent cité dans ce document.

environ 8 minutes – soit plus 25 le temps requis par notre architecture – sont nécessaires pour créer une telle machine virtuelle.

Notre architecture tire ainsi partie du fait de stocker les données des applications en réseau (le temps de montage du système de fichiers étant très peu influencé par le volume de données qui y sont stockées) et garantit un temps constant et raisonnablement faible pour le démarrage d'une tâche.

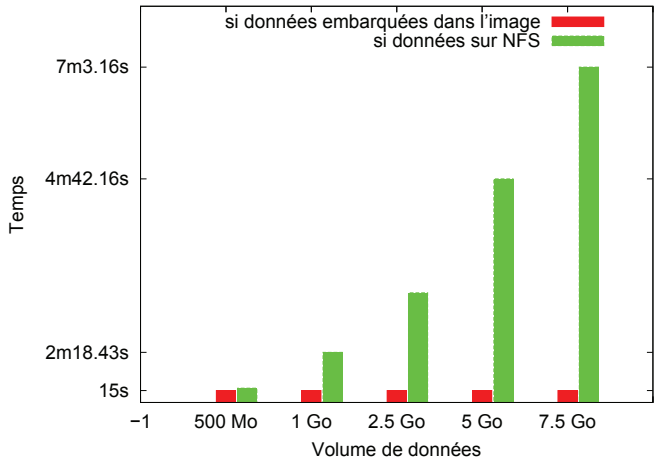

FIGURE 7.1 – Temps de création d'une machine virtuelle (DomU) selon que les données des applications sont stockées en réseau (par le biais de NFS) ou embarquées dans l'image.

7.2.1.2 Surcouts de la virtualisation

Pour évaluer les dégradations induites par la virtualisation, divers scénarios d'exécution ont été réalisés à partir des trois applications Parsec considérées (Blackscholes, Bodytrack et Freqmine) :

- Sur un nœud natif (sans aucun environnement de virtualisation), nous avons exécuté les différentes applications. Pour chaque application, nous avons fait varier le nombre d'instances (1 , 2 , 4), ainsi que le nombre de threads OpenMP actifs par instance (8 , 4 , 2) de sorte que le nombre d'instances multiplié par le nombre de threads soit au plus égal à 8 (nombre de cœurs par nœud).
- Nous avons ensuite exécuté les différentes applications à partir de machines virtuelles. Plusieurs machines virtuelles (2 ou 4) peuvent s'exécuter simultanément sur un nœud. Dans ce cas, chacune d'elles a la même configuration matérielle (nombre de cœurs virtuels ou mémoire), tandis le nombre total de cœurs virtuels ne dépasse pas 8, le nombre de cœurs disponibles sur un nœud. Pour chaque application, chaque machine virtuelle exécute une instance de l'application qui a le même nombre de threads OpenMP que de cœurs virtuels disponibles dans la machines virtuelles.

Dans nos observations, les temps observés à partir d'un nœud natif sont considérés comme les performances de référence. Elles seront comparées aux performances obtenues à partir des machines virtuelles.

Nous avons fait les constats suivants :

Blackscholes : Les performances observées à partir des machines virtuelles sont inférieures, mais assez proches de celles observées sur une machine réelle. Comme le montre les premiers groupes sur les figures 7.2-a, 7.2-b, 7.2-c et 7.2-d). Ces dégradations vont de moins de 2% (figure 7.2-a) à au plus 8%, figure 7.2-b.

Ces résultats montrent en fait qu'avec ce type d'application qui travaillent plus en mémoire, la virtualisation apporte une valeur ajoutée intéressante en permettant de mieux exploiter la puissance de calcul disponible. En effet, lorsque le nombre de processeurs et la mémoire disponibles le permettent, nous pouvons exécuter plusieurs tâches sur les nœuds. En sachant que, en plus d'avoir un bon niveau de cloisonnement entre ces tâches, les dégradations de performances induites par la virtualisation seront négligeables.

Bodytrack : Dans ce cas le surcout de la virtualisation est varie de 3% (avec une tâche unique, figure 7.2-a) à plus de 50% lorsque plusieurs tâches s'exécutent simultanément, voir par exemple la figure 7.2-c et la figure 7.2-d.

Ainsi pour ce type d'applications ayant beaucoup de besoins en entrée-sortie, nous devons éviter d'exécuter plusieurs tâches simultanément un nœud donné. Nous pouvons plutôt imaginer combiner ce type de tâche avec des tâches d'autres applications travaillant plus en mémoire, comme Blackscholes.

Freqmine : Ici les machines virtuelles présentent généralement performances proches, voire meilleures, que celles observées sur un nœud natif. En fait les dégradations lorsqu'elles existent (figure 7.2-a et figure 7.2-b) sont faibles représentant moins de 6%, notamment lors de l'exécution d'une tâche unique. Par contre, lorsque plusieurs tâches s'exécutent simultanément (figure 7.2-c et figure 7.2-d), les performances observées à partir des machines virtuelles sont supérieures à celles observées à partir d'une machine réelle.

Ce qui veut dire que pour ce type d'applications qui demandent dynamiquement beaucoup de mémoire, la virtualisation présente plus d'intérêts qu'un système natif pour exécuter simultanément plusieurs tâches sur un nœud.

7.2.2 Analyse du partage de ressources

Pour évaluer nos algorithmes par rapport aux aspects équité et utilisation efficace de ressources, nous avons pris référence sur un mécanisme de partage par découpage statique. A remarquer que lorsque $D_{seuil} = 0$, notre stratégie correspond à un découpage statique. Après chaque expérience, nous analyserons les informations collectées dans les fichiers journaux (*svmsched-core.log* et *svmsched-monitor.log*, cf. figures 6.2 et 6.3 respectivement).

Nous étudierons également l'intérêt d'estimer préalablement les besoins d'utilisation de ressources par chaque service. En effet, en faisant varié le poids affecté aux tâches longues des différents services (paramètre *weight* de la définition d'un service), nous analyserons aussi l'utilisation de ressources et le temps nécessaire pour traiter l'ensemble des tâches.

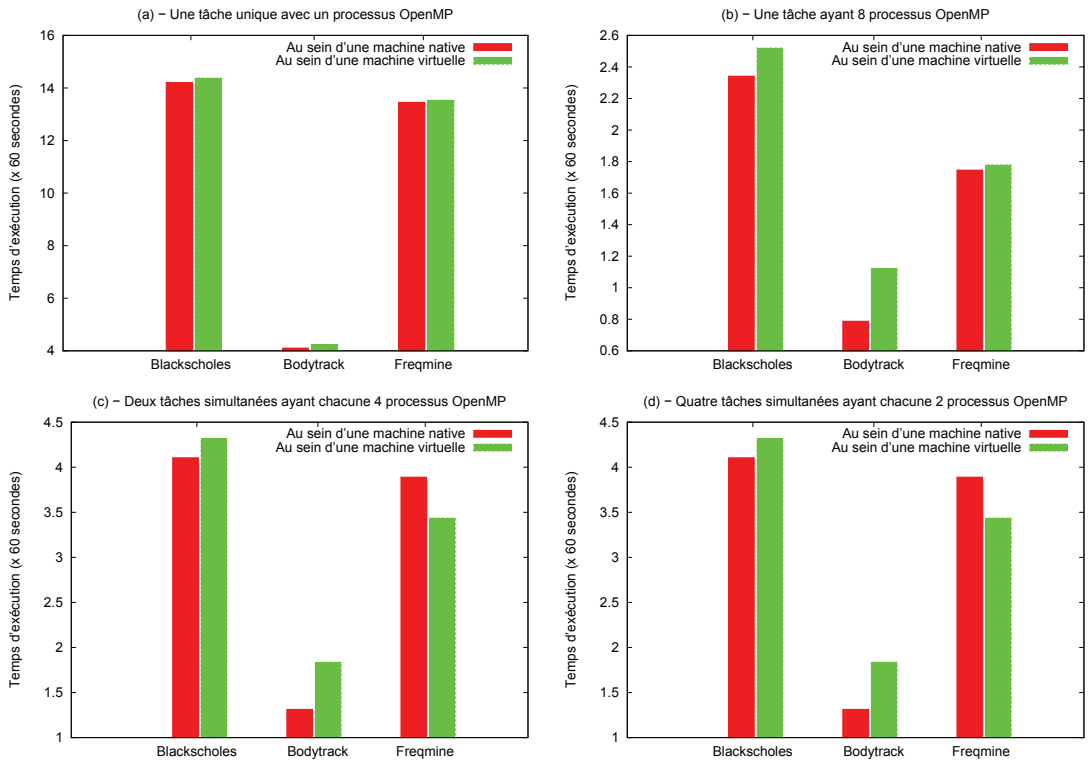

FIGURE 7.2 – Évaluation des surcouts de performances de la virtualisation

7.2.2.1 Partage statique, partage flexible : impact sur l'utilisation

Ces premiers résultats sont basés sur la charge de travail de 96 tâches, qui ont été soumises (avec des délais entre elles) sur la plateforme composée de $6 * 8 = 48$ cœurs.

Les expériences de la figure 7.3 montrent l'évolution de l'usage des ressources au cours du temps. Nous avons supposé que toutes les entreprises ont investi à la même hauteur sur la plateforme. Ainsi, lorsqu'un partage statique (figure 7.3-a) est employé, des tâches d'un service (indépendamment de leur durée) ne peuvent utiliser plus des $1/3$ de ressources en même temps. En revanche, avec notre stratégie (figure 7.3-b), cette restriction n'est appliquée qu'à des tâches longues qui ne peuvent donc utiliser plus des $1/3$ de ressources à la fois (avec D_{seuil} valant $30\,minutes$).

Dans le premier cas (figure 7.3-a), nous observons que gestionnaire SaaS est aussi en mesure de maintenir un partage rigide en assurant qu'aucun service ne puisse utiliser plus de ressources que ce qui lui est accordé. Même si, par exemple, la demande relative au service App_1 est très importante, ses tâches n'utilisent pas plus des $1/3$ de ressources qui lui sont allouées (les limites des ratios ou proportions d'usage de ressources sont marquées par les traits interrompus horizontaux). Cependant, à cause du partage rigide, une qualité importante de ressources reste sous-exploitée comme le montre les zones claires entre le ratio d'usage accordé et l'usage réel) : le taux moyen d'utilisation des ressources CPU est juste d'un peu plus de 31%.

Avec notre stratégie par contre (figure 7.3-b), nous observons que la souplesse introduite permet d'exécuter des tâches courtes pour ainsi améliorer l'utilisation globale

des ressources (les dépassements de capacités entrainés par cette souplesse sont représentés par les parties des courbes symétriques aux traits horizontaux marquant les *pseudo-limites* de ratios). Pour information, cette amélioration de l'utilisation représente en moyenne près de 17% par rapport à un partage strict. Cela conduit à réduire le temps requis pour le traitement de l'ensemble des tâches de près d'une heure. Nous pouvons alors constater que cela ne retarde pas pour autant le temps de terminaison de l'ensemble des tâches d'un service qui demande peu de ressources que ce qui lui a été accordé. Voir le service App_2 par exemple.

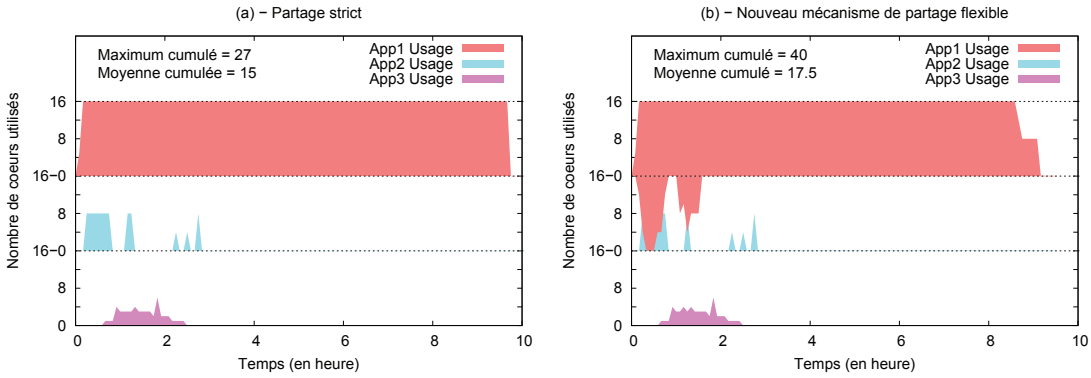

FIGURE 7.3 – Impact du mécanisme de partage de ressources sur l'utilisation et le temps global d'exécution

7.2.2.2 Intérêt du dimensionnement

A partir des résultats précédents, nous pouvons faire le constat suivant : malgré l'amélioration de l'utilisation apportée par notre mécanisme, le taux moyen d'utilisation de ressources reste encore relativement faible à environ 48%. La raison étant que les services App_2 et App_3 ont de faibles demandes, tandis que le service App_1 a une forte demande par rapport aux ratios affectés à leurs tâches longues respectives. La flexibilité introduite par la nouvelle stratégie (avec $D_{seuil} = 30\,minutes$) ne peut résoudre le problème de sous-utilisation qu'à un niveau globalement acceptable.

Ce comportement n'est pas surprenant et ces observations confortent en effet nos analyses lorsque nous mentionnions l'intérêt que chaque entreprise investisse en tenant compte de ses besoins. Ce qui nécessite de dimensionner au préalablement les besoins d'utilisation de ressources. Dans le cas présent, tout se passe comme si le fournisseur du Service App_1 avait « sous-investi »alors que ceux des Services App_2 *et* App_3 avaient « sur-investis ». Cela pourrait se prêter dans un cas réel de mutualisation à un comportement égoïste de la part du fournisseur du Service App_1.

Pour simuler une situation où chaque entreprise investit en fonction de ses besoins, nous avons supposé qu'elles ont chacune investi à hauteur du nombre de tâches liées à leur service : les poids affectés à ces différents services étaient alors de 68 (pour le service App_1), 20 (pour le service App_2) et 12 (pour le service App_3). Les expériences ont été également réalisées dans le même environnement que que précédemment, et toutes les 96 tâches ont été soumises simultanément afin de réduire la durée des expériences.

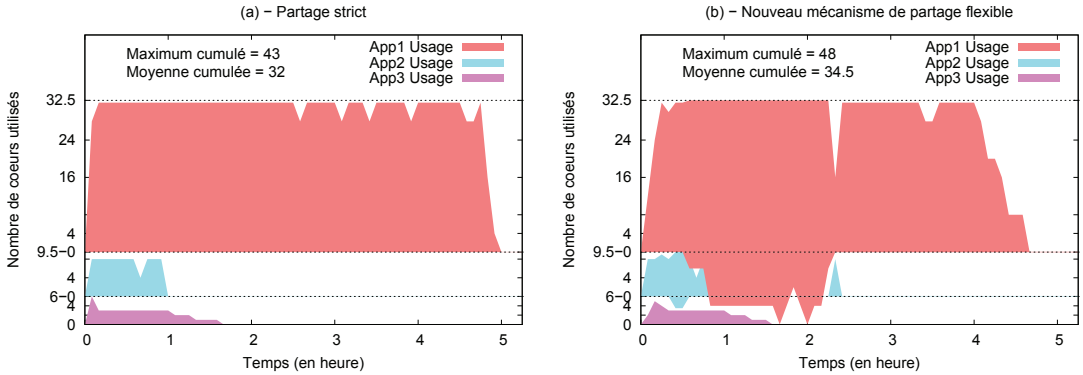

FIGURE 7.4 – Utilisation de ressources lorsque les proportions tiennent compte des besoins.

Les résultats sont présentés sur la figure 7.4. Nous observons une amélioration significative de l'utilisation, cela représente 67% en moyenne avec un mécanisme de partage strict et plus de 72% avec notre stratégie, contre 31% et 48% précédemment. Nous voyons ici qu'un dimensionnement de la plateforme par rapport aux besoins permet à la mutualisation de joue son rôle, c'est-à-dire d'apporter un volet fédération de ressources pour avoir plus de puissance de calcul dont l'utilisation est optimisée par complémentarité entre les services.

7.2.3 Passage à l'échelle

Pour étudier ces problématiques, les expériences que nous analyserons dans cette partie ont été réalisées avec la charge de travail de 1000 tâches, qui ont été soumises simultanément sur la seconde plateforme composée de $20 * 8 = 160$ cœurs. Ici, nous ne présenterons que les résultats observés avec notre stratégie de gestion de ressources (en prenant $D_{seuil} = 15\,minutes$ comme indiqué à la page 91). Dans les expériences, nous avons en outre supposé que chaque fournisseur a investi en tenant compte de ses besoins (proportion de tâches relatives à son service, 50% pour le service App_1, 30% pour le service App_1, 20% pour le service App_3).

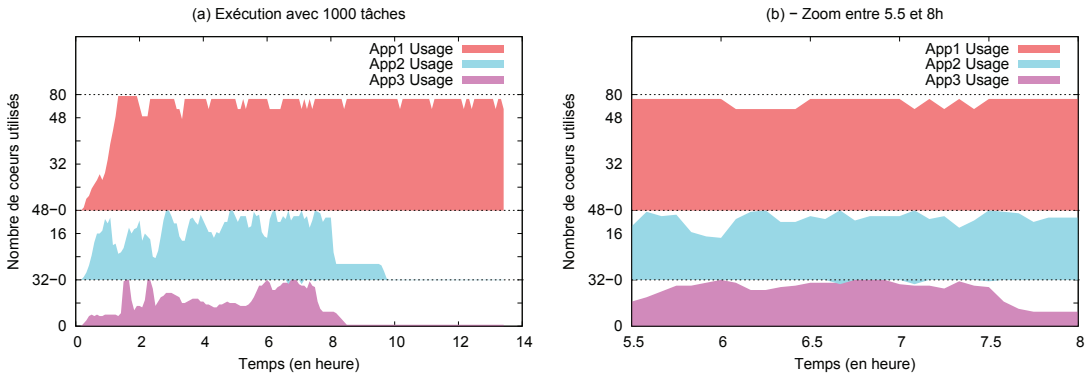

FIGURE 7.5 – Exécution avec une charge de travail de 1000 tâches sur une plateforme de $20 * 8 = 160$ cœurs.

D'un point de vue performance, nous avons observé – toujours à partir des informations collectées dans les fichiers journaux – q'un temps de 8 secondes ont été néces-

saires pour l'admission (analyse, validation et sauvegarde) de l'ensemble des tâches 1000. Ce qui correspond à une moyenne de 8 microsecondes (μs) par tâche, en rappel l'admission d'une tâche est traitée par un processus indépendant (agent *validateur*) spécialement crée à la volée à cet effet.

Ce surcout de 8μs observé semble faible comparé au temps de traitement d'une tâche. De manière générale, le temps requis pour l'admission de l'ensemble des tâches montre que notre système est capable de supporter une forte charge avec des performances intéressantes.

Concernant la mutualisation, la figure 7.5 nous suggère de nouveaux commentaires quant à la problématique de dimensionnement de la plateforme et surtout, du choix du paramètre D_{seuil}. En effet, en choisissant $D_{seuil} = 15\,minutes$, très peu de tâches semblent être considérées comme tâches courtes. C'est-à-dire des tâches ayant chacune une durée inférieure ou égales à D_{seuil}. Cette figure montre en réalité que nous ne tirons pas pleinement partie de notre stratégie pour maximiser l'utilisation de ressources : ce n'est qu'entre 6 et 7 heures (mieux perceptible sur la figure 7.5-b) que des tâches courtes (toutes liées au service App_2) tirent partie de la stratégie.

Ainsi cet exemple montre le caractère essentiel du paramètre D_{seuil} dans la stratégie de mutualisation. Ici, la mutualisation semble avoir peu d'intérêt car tout se passe comme si chacune des entreprises disposait de ressources qui lui sont exclusivement dédiées.

7.3 Discussions

De manière générale, nous constatons que les résultats observés consolident nos choix. Que ce soit du point de vue de l'architecture que de celui du mécanisme de gestion de ressources, ces résultats sont conformes aux objectifs annoncés :

- Le mécanisme de gestion de ressources permet de répondre convenablement au problème de partage proportionnel et d'utilisation efficace de ressources dans un contexte de ressources mutualisées. Remarquons en plus que, dans nos expériences, nous avons supposé que toutes les tâches sont exécutées en mode production. C'est-à-dire sans préemption et avec des règles d'allocation de ressources contraignantes. Or, notre système supporte également un mode best-effort (préemptif) où une tâche en attente peut être exécutée sur n'importe quelle ressource libre. Ce qui veut dire qu'en pratique, lorsque des tâches internes aux fournisseurs de services seront exécutées dans ce mode, l'utilisation de ressources serait encore plus accrue.
- L'architecture virtualisée qui repose sur des machines virtuelles ultra-légères, dont les applications accèdent aux données via le réseau, permet d'avoir une dynamicité et une automatisation remarquable. Leur création, leur configuration et leur démarrage sont réalisées à la volée, et de manière transparente, au démarrage d'une tâche. Nous l'avions montré, le surcout induit est négligeable comparé au temps d'exécution de beaucoup de tâches. Sans compter le fait que ce mécanisme apporte une dynamicité et une automatisation pour éviter des interactions homme-machine avant l'exécution d'une tâche.

- En outre, les machines virtuelles permettent d'optimiser l'utilisation des nœuds grâce à des consolidations avec de bons niveaux d'isolation. Nous avons constaté que les dégradations de performances induites par la virtualisation sont acceptables. Dans certains cas, nous avons même observé que des exécutions multiples via des machines virtuelles s'avèrent plus intéressantes que sur un système natif.
- Enfin, nous avons fait une évaluation de performances de machines virtuelles Xen qui actualise les précédents résultats (cf. chapitre 2, page 37) basés sur des versions antérieures de Xen. De plus, contrairement à ces études, ces résultats sont basés sur un cas particulier d'architecture où les applications s'exécutant au sein des machines virtuelles accèdent aux données via un système de stockage réseau NFS .

Ainsi se conclut ce chapitre qui valide nos contributions, le chapitre suivant conclut l'ensemble de cette thèse et présente des perspectives pour des travaux futurs.

Conclusion et Perspectives 8

Dans ce chapitre de conclusion, nous reviendrons d'abord sur les objectifs et les contributions. La section 8.2 présentera des pistes de nos travaux futurs. Le chapitre se terminera, section 8.3, par des remarques et commentaires personnels sur les travaux et le contexte de la thèse.

8.1 Contributions

Durant cette thèse, nous nous sommes intéressés au problème de gestion de ressources sur une grappe mutualisée pour des services de calcul intensif à demande. Dans ce contexte où les services appartiennent à plusieurs, mais en nombre limité d'entreprises, nous visions à traiter les problématiques suivantes :

Gestion dynamique et cloisonnement de tâches : Il s'agissait de répondre à des problématiques de prise en charge et de traitement des requêtes de service de manière automatique et transparente pour les clients.

Pour atteindre cet objectif, nous avons défini une architecture où l'exécution d'une tâche est réalisée au sein d'une machine virtuelle qui est créée, personnalisée et déployée à la volée pour exécuter la tâche. Conçue avec un souci de réutilisation, cette architecture s'appuie sur un gestionnaire d'infrastructure virtuelle pour gérer le cycle de vie des machines virtuelles de manière efficace sur la grappe. Notre architecture peut ainsi supporter divers hyperviseurs tels que Xen, KVM, VMware ESXi/ESX ou Microsoft Hyper-V, de manière à réutiliser à différents niveaux des fonctionnalités de virtualisation déjà testées et éprouvées dans des outils existants. Par conséquent, cela nous a permis de nous focaliser sur les fonctionnalités spécifiques pour nos objectifs d'exploitation. Ces nouvelles fonctionnalités ont été intégrées dans une composante spéciale de l'architecture : le gestionnaire SaaS, qui fournit l'abstraction et des mécanismes appropriés pour atteindre nos objectifs.

Motivé par un souci de performances, de cout, mais surtout d'interopérabilité, via notamment l'ouverture des codes et des standards, nous avons développé un prototype en nous basant sur OpenNebula et Xen. Nous avons vu que cette architecture nécessite un faible couplage avec OpenNebula, les interactions avec ce dernier n'étant réalisées que par le biais des interfaces de haut niveau (APIs XML-RPC). Ainsi, tant que ces APIs restent inchangées, les composantes tels que le gestionnaire SaaS, propres à l'architecture, ne sont pas dépendantes des changements au niveau de ses autres composantes (une évolution du code par exemple).

Partage proportionnel et utilisation efficace de ressources : Nous cherchions à définir un mécanisme de gestion de ressources pour garantir deux objectifs conflictuels : partager proportionnellement les ressources de la plateforme entre les différents services tout en maximisant leur utilisation, sachant que chacune des entreprises peut investir à différente proportion sur la plateforme.

Nous avons établi des algorithmes et des mécanismes de gestion de ressources qui offrent de propriétés intéressantes pour une telle mutualisation de ressources. Ces outils s'appuient également sur la virtualisation qui, outre le fait de simplifier la gestion dynamique de tâche, permet de mieux partitionner les ressources via le multiplexage et le cloisonnement de tâches sur un nœud physique. Cette virtualisation apporte une souplesse qui permet d'optimiser l'utilisation de ressources, notamment en permettant la prise en charge de l'exécution des tâches non-liées à des clients en mode best-effort, c'est-à-dire avec possibilité de suspension et de reprise, sur des ressources inutilisées.

Prototypage, évaluation et validation : Cette thèse ayant des motivations industrielles, nous avons développé une preuve de concept pour valider nos contributions.

Un prototype (SVMSched) du gestionnaire SaaS a été développé puis évalué expérimentalement. Les résultats montrent que nos stratégies et algorithmes satisfont les objectifs d'exploitation fixés au départ.

Plus spécifiquement, nos travaux ont fait l'objet d'une évaluation avec des retours très positifs dans le cadre de la revue finale du projet Ciloe.

8.2 Perspectives

A l'état actuel, notre système doit encore être amélioré avant d'être prêt pour être intégré dans un environnement de nuage de calcul en production. A plus ou moins court terme, les pistes d'amélioration visent les aspects suivants :

Prédiction des temps d'exécution des tâches : Notre stratégie de partage de ressources est basée sur le fait que les durées d'exécution des différentes tâches sont connues à l'avance. Dans nos expériences, la charge de travail permettait d'obtenir ces informations. Cependant, dans un environnement de production où de telles informations ne seraient pas toujours disponibles a priori, il serait intéressant que le système puisse les prédire de manière dynamique et transparente.

En effet, étant face à des applications bien maitrisées (par leur éditeur), les facteurs influençant les performances pourront être étudiés finement pour établir un modèle de performance. Ayant déjà fait l'objet d'une étude préliminaire [52], nous entrevoyons la réalisation de cet objectif à court terme. Dans cette étude, nous montrons à partir d'informations simples sur les données en entrée, comment dimensionner des structures de données dynamiques qui influencent les temps d'exécution d'un code de l'application Jivaro (voir page 16). Mais, de manière générale, le modèle de prédiction pourrait être établi à partir d'échantillons de temps d'exécution issues de traitements antérieurs. Et pour plus de précisions, le modèle pourrait être enrichi avec des méthodes de type apprentissage par renforcement [119].

Modèle économique ou de facturation : A moyen terme, nous devons étudier le modèle économique pour déterminer les clients auxquels ce type de service pourrait s'adresser. En réalité, il se posera un problème de compromis entre la sensibilité des données que le client soumet et le bénéfice escompté du service. Par exemple, en cas de fuite des données, une entreprise cliente risque sa compétitivité. Un tel service ciblerait typiquement des clients dont les budgets de fonctionnement ne permettraient pas de supporter l'acquisition de licence permanente du logiciel sous-jacent.

D'un point de vue commercial, il serait également intéressant de définir un modèle de facturation adapté pour des services de calcul intensif. Les modèles existants, où les services sont essentiellement basés sur des applications collaboratives ou de bureautique, ne correspondent pas. Au lieu de contrats ou souscriptions sur la durée comme dans ces cas, notre modèle de facturation ne prendrait en compte que l'utilisation *ponctuelle* du matériel et du logiciel de calcul. Ce modèle pourrait tirer partie de la notion de bail que nous avons défini, en considérant notamment le temps et la quantité de ressources requis pour un traitement.

Dimensionnement de la plateforme : Enfin, il serait intéressant à plus long terme d'approfondir les questions du dimensionnement avant la mise en œuvre d'une telle infrastructure mutualisée. Ce dimensionnement visera à mettre en place une plateforme qui répond convenablement aux besoins des différentes entreprises. Notamment, cela doit répondre à des questions liées aux caractéristiques matérielles, à la consommation énergétique, ou à la définition des paramètres tels que D_{seuil} (cf. page 63) influençant le partage et l'utilisation de ressources.

8.3 Conclusions et commentaires personnels

Dans un cadre de collaboration académique et industrielle, j'ai travaillé durant cette thèse à proposer et implémenter une solution pour mutualiser des ressources de calcul. Bien que le problème ait été posé pour des acteurs intéressés par des nuages de type SaaS, nous avons montré [51] que la solution apportée pourrait être étendue dans un contexte PaaS.

D'un point de vue scientifique, les contributions ont donné lieu à des communications dans des conférences nationales et internationales. Certaines de ces contributions ont fait l'objet d'un chapitre du livre *Cloud Computing and Services Science* à paraitre chez Springer-Verlag. Mes travaux de recherche, de développement et d'expérimentation durant cette thèse m'ont valu un prix scientifique ($1er$ Prix Challenge Grid'5000).

De manière générale, cette thèse a permis à des acteurs académiques de s'intéresser à des problématiques dont les solutions pourront être développées et commercialisées dans le court ou moyen terme. Aussi, cela a été un cadre favorable à la compétitivité des industriels qui ont pu bénéficier d'outils ou de méthodologies de recherche propices à l'innovation.

En ce qui me concerne, ce fut une expérience enrichissante. Pour un étudiant en thèse, travailler en interface sur des problématiques industrielles et scientifiques constitue un levier stratégique qui permet soit de rester dans le milieu académique, soit de basculer dans l'industrie.

Annexes et Index

TABLE 1 – Caractéristiques de quelques hyperviseurs

Outil	Types de virtualisation	Interopérabilité	Autres Détails
Xen, XenServer, XCP	Virtualisation complète, paravirtualisation, virtualisation matérielle, Support Virtio	Libvirt	Open source et gratuit, extensions commerciales (ex. XenServer), support communauté et commercial (extensions)
KVM	Virtualisation complète et Virtualisation matérielle + Support Virtio	Compatible Libvirt	Open source et gratuit, support communauté
VMware ESXi/ESX	Virtualisation complète, paravirtualisation, virtualisation matérielle	Compatible Libvirt, Support OVF, API SOAP propre	Propriétaire, versions commerciales et gratuite – limitée, support commercial
Hyper-V	Virtualisation complète, paravirtualisation, virtualisation matérielle	Compatible Libvirt	Propriétaire, versions commerciale et gratuite – limitée, support commercial

TABLE 2 – Caractéristiques de quelques gestionnaires d'infrastructures distribuées virtualisées

Outil	**Hyperviseurs Supportés**	**Ordonnancement de machines virtuelles (MVs)**	**Interopérabilité**	**Autres détails**
Eucalyptus	Xen, KVM, VMware ESXI/ESX	**Inexistant**, placement immédiat de MVs (glouton, roundrobin)	APIs compatibles Amazon EC2/EBS/S3	Open source et gratuit en version limitée
Enomaly ECP	Xen, KVM, VMware ESXi/ESX	**Inexistant**, placement immédiat de MVs (glouton, roundrobin)	APIs REST propres	Propriétaire et commercial
Nimbus	Xen, KVM	Basique, provisionnement immédiat de MVs	APIs compatibles Amazon EC2/S3	Open source et gratuit, destiné à la science
OpenNebula	Xen, XenServer, XCP, KVM, WMware ESXi/ESX, Hyper-V	**Effectif**, placement de MVs géré par un ordonnanceur personnalisable	APIs compatibles Amazon EC2, XML-RPC (propre) et OCCI	Open source et Gratuit, Modulaire et facilement personnalisable avec des outils via ses APIs XML-RPC et OCCI.
OpenStack	Xen, XenServer, XCP, KVM, WMware ESXi/ESX, Hyper-V	**Effectif**, placement de MVs géré par un ordonnanceur personnalisable	APIs compatibles Amazon EC2/EBS/S3 , APIs REST propres	Open source et gratuit, support communauté, extensions commerciales non matures
VMware vSphere	VMware ESXi/ESX	**Inexistant**, placement immédiat de MVs ou via DRS qui prend en compte l'équilibrage de charge et la haute disponibilité	API REST et SOAP propres	Propriétaire et Gratuit en version limitée à 60 jours,

Index

Backfilling, 28
Bail, définition, 61

Confinement d'application, 34
Contextualisation, 41, 59

D_{seuil}, 63, 64
DRS, VMware, 42

Enomaly ECP, 40
Eucalyptus, 39

Fair-share, 29

Gestionnaire d'infrastructure virtuelle, 35
Grid'5000, 20

Haizea, 39, 40

Jivaro, 16
Journal de supervision, 84
Journal des événements, 81

Libvirt, 37

Nimbus, 39

OCCI, 38
OpenNebula, 40
OpenStack, 41

Paravirtualisation, 32

Requête, définition, 10

Service, définition, 10

Tâche, définition, 10
    best-effort, 61
    de production ou de service, 61

Validateur, agent, 81
Virtualisation assistée par le matériel, 34
Virtualisation complète, 32
Virtualisation hybride, 34
vSphere, VMware, 41

Références et Bibliographie

Bibliographie

[1] Amazon Web Services. http ://aws.amazon.com/.

[2] Apps.Gov. https ://www.apps.gov/cloud/main/home.do ?

[3] BSD Jail. http ://www.freebsd.org/doc/handbook/jails.html.

[4] cgroups. http ://www.kernel.org/doc/Documentation/cgroups/cgroups.txt.

[5] Cloud Scheduler. http ://cloudscheduler.org/.

[6] Cooperative Linux . http ://www.colinux.org/.

[7] cpusets. http ://www.kernel.org/doc/Documentation/cgroups/cpusets.txt.

[8] DropBox. http ://www.dropbox.com/.

[9] Enomaly. http ://enomaly.com.

[10] Force.com. http ://www.salesforce.com/platform/.

[11] Globus. http ://www.globus.org/.

[12] GoGrid Cloud Hosting. http ://www.gogrid.com/.

[13] Google App Engine. http ://code.google.com/appengine/.

[14] Google Apps. http ://www.google.com/apps/.

[15] Grid'5000. http ://www.grid5000.fr/.

[16] Linux-VServer. http ://linux-vserver.org/.

[17] LoadLeveler. http ://ibm.com/systems/software/loadleveler/.

[18] Manuel de référence Linux/UNIX pour chroot.

[19] Microsoft Azure Platform. http ://www.microsoft.com/windowsazure/.

[20] OCCI. http ://occi-wg.org/.

[21] Office Web Apps. http ://office.microsoft.com.

[22] OpenStack. http ://openstack.org/.

[23] OpenVZ. http ://wiki.openvz.org.

[24] PBS Works. http ://pbsworks.com/.

[25] Rackspace Hosting. http ://www.rackspace.com/.

[26] Salesforce. http ://www.salesforce.com/.

[27] System Administration Guide : Oracle Solaris Containers-Resource Management and Oracle Solaris Zones. http ://docs.oracle.com/cd/E19455-01/817-1592/.

[28] User-Mode Linux. http ://user-mode-linux.sourceforge.net/.

[29] XaaS. http ://searchcloudcomputing.techtarget.com/definition/XaaS-anything-as-a-service.

[30] XML-RPC. http ://www.xmlrpc.com/.

[31] AMD. Amd64 virtualization codenamed asia pacific technology : Secure virtual machine architecture reference manual. (Publication No. 33047, Revision 3.01), May 2005.

[32] David P. Anderson. Boinc : A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[33] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home : an experiment in public-resource computing. *Commun. ACM*, 45 :56–61, November 2002.

[34] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53 :50–58, April 2010.

[35] Abbas Asosheh and Mohammad Hossein Danesh. Comparison of OS level and hypervisor server virtualization. In *Proceedings of the 8th conference on Systems theory and scientific computation*, pages 241–246. World Scientific and Engineering Academy and Society (WSEAS), 2008.

[36] Lee Badger, Tim Grance, Robert Patt-Corner, and Jeff Voas. DRAFT Cloud Computing Synopsis and Recommendations. Technical report, U.S. Department of Commerce Gary Locke, Secretary National Institute of Standards and Technology Patrick D. Gallagher, Director, 2011-08-14 14 :40 :29.

[37] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, Alex Ho, R. Neugebauer, Ian Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03 : Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177. ACM, 2003.

[38] Fran Berman, Geoffrey Fox, and Anthony J. G. Hey. *Grid Computing : Making the Global Infrastructure a Reality*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[39] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[40] J Blazewicz, K. Ecker, B. Plateau, and D. Trystram. *Handbook on Parallel and Distributed Processing*. Springer, 2000.

[41] Brett Bode, David M. Halstead, Ricky Kendall, Zhou Lei, and David Jackson. The portable batch scheduler and the maui scheduler on linux clusters. In *Proceedings of the 4th annual Linux Showcase & Conference*, pages 27–27. USENIX Association, 2000.

[42] Sotomayor Borja, Keahey Kate, Foster Ian, and Freeman Tim. Enabling cost-effective resource leases with virtual machines. In *Hot Topics session in ACM/IEEE International Symposium on High Performance Distributed Computing*, 2007.

[43] David R. Butenhof. *Programming with POSIX threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[44] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for Service-Oriented grid computing. 2001.

[45] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic Models for Resource Management and Scheduling in Grid Computing. *The Journal of Concurrency and Computation : Practice and Experience (CCPE)*, 2002.

[46] Rajkumar Buyya. Grid economy comes of age : Emerging gridbus tools for service-oriented cluster and grid computing. In *Proceedings of the Second International Conference on Peer-to-Peer Computing*, P2P '02, pages 13–, Washington, DC, USA, 2002. IEEE Computer Society.

[47] Rajkumar Buyya, Suraj Pandey, and Christian Vecchiola. Cloudbus toolkit for market-oriented cloud computing. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 24–44, Berlin, Heidelberg, 2009. Springer-Verlag.

[48] Rajkumar Buyya, Chee S. Yeo, and Srikumar Venugopal. Market-Oriented Cloud Computing : Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. August 2009-09-08 11 :57 :39.

[49] Nicolas. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and Olivier Richard. A batch scheduler with high level components. In *Cluster computing and Grid*, 2005.

[50] P. Chacin, X. León, R. Brunner, F. Freitag, and L. Navarro. Core services for grid markets. In *The CoreGRID Symposium (CGSYMP 2008)*, August 2008.

[51] Rodrigue Chakode. SVMSched : a tool to enable On-demand SaaS and PaaS on top of OpenNebula. *OpenNebula Blog*, http ://blog.opennebula.org/ ?p=1646, June 2011.

[52] Rodrigue Chakodé, Jean-Francois Mehaut, and Francois Charlet. High Performance Computing on Demand : Sharing and Mutualizing Clusters. In *24th IEEE International Conference on Avanced Information Networking and Applications*, pages 126 – 133, Perth, Australia, 2010.

[53] Rodrigue Chakode, Jean-Francois Méhaut, and Blaise-Omer Yenke. *Cloud Computing and Services Science*, chapter 14. Springer-Verlag, 30 Avril 2012.

[54] Rodrigue Chakodé and Blaise Yenke. Utilisation des machines virtuelles comme support de services de calcul à la demande. In *Rencontres Francophones du Parallélisme (Renpar'20)*, Saint-Malo, France, may 2011.

[55] Rodrigue Chakodé, Blaise Yenke, and Jean-Francois Mehaut. Resource Management of Virtual Infrastructure for On-demand SaaS Services. In *The 1st International Conference on Cloud Computing and Services Science*, pages 352–361, Noordwikerhout, Netherlands, may 2011.

[56] Robit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon. *Parallel programming in OpenMP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[57] Brad Chen. Research Areas of Interest : Building scalable, robust cluster applications. http ://googleresearch.blogspot.com/2010/01/research-areas-of-interest-building.html, January 27, 2010.

[58] Brent N. Chun and David E. Culler. REXEC : A Decentralized, Secure Remote Execution Environment for Clusters. In *CANPC '00 : Proceedings of the 4th International Workshop on Network-Based Parallel Computing*, pages 1–14. Springer-Verlag, 2000.

[59] T. Cioara, I. Anghel, I. Salomie, G. Copil, D. Moldovan, and A. Kipp. Energy aware dynamic resource consolidation algorithm for virtualized service centers based on reinforcement learning. In *Parallel and Distributed Computing (ISPDC), 2011 10th International Symposium on*, pages 163 –169, july 2011.

[60] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter 16. MIT Press, 2009.

[61] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[62] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao. Quantitative comparison of xen and kvm. In *Xen summit*. USENIX association, June 2010-02-16 16 :18 :55.

[63] Neumann Dirk, Jochen Stößer, Arun Anandasivam, and Nikolay Borissov. Sorma - building an open grid market for grid resource allocation. In *GECON*, pages 194–200, 2007.

[64] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2 :115–150, May 2002.

[65] Message P Forum. Mpi : A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[66] Ian Foster and Carl Kesselman. *The Grid 2 : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[67] John Gehl. Question time : Napster. *Ubiquity*, 2000, July 2000.

[68] Nelson M. Gonzalez, Charles Miers, Fernando F. Redigolo, Marcos A. Simplício Jr., Tereza Cristina M. B. Carvalho, Mats Näslund, and Makan Pourzandi. A taxonomy model for cloud computing services. In *CLOSER*, pages 56–65, 2011.

[69] Georg Hager and Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010.

[70] Georg Hager and Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010.

[71] Michael T. Heath. *Scientific Computing : An Introductory Survey*. McGraw-Hill ., New York, NY, USA, 2002.

[72] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy : a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 41–50, New York, NY, USA, 2009. ACM.

[73] Hyper-V Server. http ://microsoft.com/hyper-v-server/.

[74] Intel. Virtualizing the Data Center. www.intel.com/it/pdf/virtualizing-the-data-center.pdf, March 2007.

[75] Intel. Implementing On-Demand Services Inside the Intel IT Private Cloud. http ://www.intel.com/content/dam/doc/white-paper/intel-it-private-cloud-on-demand-services-paper.pdf, October 2010.

[76] Intel Corporation. Intel Virtualization Technology. *Intel Technology Journal*, 10(3), August 2006.

[77] Menken Ivanka. *SaaS - The Complete Cornerstone Guide to Software as a Service Best Practices Concepts, Terms, and Techniques for Successfully Planning, Implementing and Managing SaaS Solutions*. Emereo Pty Ltd, 2008.

[78] Sam Johnston. Taxonomy : The 6 layer Cloud Computing Stack. http ://samj.net/2008/09/taxonomy-6-layer-cloud-computing-stack.html, 18 Semtember 2008.

[79] Tim Jone. Linux virtualization and pci passthrough. http ://www.ibm.com/-developerworks/linux/library/l-pci-passthrough/.

[80] Kay, J. and Lauder, P. A fair share scheduler. *Commun. ACM*, 31(1) :44–55, January 1988.

[81] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces : Achieving quality of service and quality of life in the grid. *Sci. Program.*, 13 :265–275, October 2005.

[82] Tony Kontzer. Cloud Computing : Anything as a Service. http ://www.cioinsight.com/c/a/Strategic-Tech/Cloud-Computing-Anything-as-a-Service/, 05 August 2008.

[83] Ruby Krishnaswamy, Leandro Navarro, and Vladimir Vlassov. A democratic grid : Collaboration, sharing and computing for everyone. In *eChallenges'08 Collaboration and the Knowledge Economy : Issues, Applications, Case Studies*, volume 1. IOS Press, IOS Press, 10/2008 2008.

[84] Vipin Kumar. *Introduction to Parallel Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[85] KVM. http ://linux-kvm.org/.

[86] Libvirt Virtualization API. http ://libvirt.org/.

[87] P. Lin, A. MacArthur, and J. Leaney. Defining autonomic computing : a software engineering perspective. In *Software Engineering Conference, 2005. Proceedings. 2005 Australian*, pages 88 – 97, march-1 april 2005.

[88] David Linthicum. Defining the Cloud Computing Framework. http ://cloudcomputing.sys-con.com/node/811519, 18 January 2009.

[89] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[90] Jiuxing Liu, Wei Huang, Bulent Abali, and Dhabaleswar K. Panda. High performance vmm-bypass i/o in virtual machines. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 3–3. USENIX Association, 2006.

[91] Paul Marshall, Kate Keahey, and Tim Freeman. Improving utilization of infrastructure clouds. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 205–214, Washington, DC, USA, 2011. IEEE Computer Society.

[92] Dejan S. Milojičić, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. *ACM Comput. Surv.*, 32 :241–299, September 2000.

[93] Anders Moller and Michael I Schwartzbach. *An introduction to XML and Web technologies*. Addison-Wesley, 2006.

[94] Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6) :529–543, 2001.

[95] Jun Nakajima and Asit K. Mallick. Hybrid-virtualization-enhanced virtualization for linux. In *Proceedings of the Linux Symposium*, pages 87 – 96, Ottawa, June 2007.

[96] Henrik F. Nielsen, Noah Mendelsohn, Jean J. Moreau, Martin Gudgin, and Marc Hadley. SOAP version 1.2 part 1 : Messaging framework. W3C recommendation, W3C, June 2006-08-28 21 :35 :17.

[97] A. Yoo nnd M. Jette and M. Grondona. SLURM : Simple Linux Utility for Resource Management. In *ClusterWorld Conference and Expo*, pages 44–60, San Jose, California, June 2003.

[98] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus open-source cloud-computing system. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, volume 0, pages 124–131. IEEE, 2009-09-03 10 :29 :48.

[99] Masato Oguchi. Research works on cluster computing and storage area network. In *ICUIMC '09 : Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, pages 366–375. ACM, 2009.

[100] E.R. Olsen. Transitioning to software as a service : Realigning software engineering practices with the new business model. In *Service Operations and Logistics, and Informatics, 2006. SOLI '06. IEEE International Conference on*, pages 266–271, June 2006.

[101] Blaise Omer Yenke, Jean-Francois Mehaut, Jean Michel Nlong II, and Rodrigue Chakodé. Integrating Deadline-Constrained Checkpointing in a Batch Scheduler for Dynamic Environments. In *Annual International Conference on Software Engineering (SE'2010)*, Thailand, 2010.

[102] Open Virtualization Format. http ://www.dmtf.org/standards/ovf/.

[103] David A. Patterson and John L. Hennessy. *Computer organization and design : the hardware/software interface*. Morgan Kaufmann, Amsterdam, Boston, 4th edition, 2009.

[104] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data : Parallel analysis with sawzall. *Sci. Program.*, 13 :277–298, October 2005.

[105] Dhanji R. Prasanna. *Dependency Injection*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2009.

[106] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking : Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, July 1998.

[107] R. Rivest. The md5 message-digest algorithm, 1992.

[108] Benny Rochwerger, David Breitgand, Eliezer Levy, Alex Galis, Kenneth Nagin, Ignacio M. Llorente, Ruben Montero, Yaron Wolfsthal, Erik Elmroth, Juan Caceres, Muli Ben-Yehuda, Wolfgan Emmerich, and Fermin Galan. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4), 2009.

[109] G. Runger and M. Schwind. Cache optimization for mixed regular and irregular computations. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –8, april 2008.

[110] Rusty Russell. virtio : towards a de-facto standard for virtual i/o devices. *SIGOPS Oper. Syst. Rev.*, 42 :95–103, July 2008.

[111] Jose Carlos Sancho, Fabrizio Petrini, Kei Davis, Roberto Gioiosa, and Song Jiang. Current practice and a direction forward in checkpoint/restart implementations for fault tolerance. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 18 - Volume 19*, IPDPS '05, pages 300.2–, Washington, DC, USA, 2005. IEEE Computer Society.

[112] Jason Sanders and Edward Kandrot. *CUDA by Example : An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition, 2010.

[113] Peter Sempolinski and Douglas Thain. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In *IEEE International Conference on Cloud Computing Technology and Science*, 2010.

[114] Jahanzeb Sherwani, Nosheen Ali, Nausheen Lotia, Zahra Hayat, and Rajkumar Buyya. Libra : a computational economy-based job scheduling system for clusters. *Softw. Pract. Exper.*, 34(6) :573–590, 2004.

[115] Jim Smith and Ravi Nair. *Virtual Machines : Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[116] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13 :14–22, September 2009.

[117] Thomas Sterling, Donald J. Becker, Daniel Savarese, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer. Beowulf : A parallel workstation for scientific computation. In *In Proceedings of the 24th International Conference on Parallel Processing*, pages 11–14. CRC Press, 1995.

[118] V. S. Sunderam. Pvm : a framework for parallel distributed computing. *Concurrency : Pract. Exper.*, 2 :315–339, November 1990.

[119] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[120] Mark Turner, David Budgen, and Pearl Brereton. Turning Software into a Service. *Computer*, 36(10) :38–44, 2003.

[121] Luis M. Vaquero, Luis Rodero-M., Juan Caceres, and Maik Lindner. A break in the clouds : towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1) :50–55, 2009.

[122] VMware. A Performance Comparison of Hypervisors. http ://www.vmware.com/resources/techresources/711, January 31, 2007.

[123] VMware. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. http ://www.vmware.com/resources/techresources/1008, November 10, 2007.

[124] VMware vSphere. http ://vmware.com/products/vsphere/.

[125] John Paul Walters, Vipin Chaudhary, Minsuk Cha, Salvatore Guercio Jr., and Steve Gallo. A comparison of virtualization technologies for hpc. In *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications*, pages 861–868, Washington, DC, USA, 2008. IEEE Computer Society.

[126] Aaron Weiss. Computing in the clouds. *netWorker*, 11(4) :16–25, 2007.

[127] Craig D. Weissman and Steve Bobrowski. The design of the force.com multitenant internet application development platform. In *SIGMOD '09 : Proceedings of the 35th SIGMOD international conference on Management of data*, pages 889–896. ACM, 2009.

[128] XenSource. A Performance Comparison of Commercial Hypervisors. http ://www.cc.iitd.ernet.in/misc/cloud/XenExpress.pdf, 2007.

[129] Blaise Omer Yenke, Jean-Francois Mehaut, and Maurice Tchuente. Scheduling of computing services on intranet networks. *IEEE Trans. Serv. Comput.*, 4 :207–215, July 2011.

[130] L. Youseff, M. Butrico, and D. Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1 –10, nov. 2008.

[131] Zhenxing Zhao, Congying Gao, and Fu Duan. A survey on autonomic computing research. In *Computational Intelligence and Industrial Applications, 2009. PACIIA 2009. Asia-Pacific Conference on*, volume 2, pages 288 –291, nov. 2009.