Ordonnancement sous contraintes de qualité de service dans les clouds

Tom Guérout

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)*

Présentée et soutenue le *05/12/2014* **par :**

Tom GUÉROUT

Ordonnancement sous contraintes de Qualité de Serivce dans les Clouds

---

JURY

Pascal BOUVRY	Université de Luxembourg	Rapporteur
Jean-Marc MENAUD	École des Mines de Nantes	Rapporteur
Jean-François MÉHAUT	Université de Grenoble 1	Examinateur
Christophe CHASSOT	INSA Toulouse	Examinateur
Thierry MONTEIL	LAAS-CNRS	Directeur de thèse
Georges DA COSTA	IRIT	Directeur de thèse

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture
Unité de Recherche :
LAAS - CNRS (UPR 8001) et IRIT (UMR 5505)
Directeur(s) de Thèse :
Georges DA COSTA et *Thierry MONTEIL*
Rapporteurs :
Pascal BOUVRY et *Jean-Marc MENAUD*

Remerciements

Mes premiers remerciements sont adressés à mes deux directeurs de Thèse, Thierry Monteil et Georges Da Costa, respectivement du LAAS et de l'IRIT. Georges et Thierry m'ont été d'un support indispensable durant ce doctorat. En partageant vos expériences et vos connaissances, toujours avec des mots justes, vous avez su me soutenir et me motiver durant ces trois années. Ce co-encadrement entre le LAAS et l'IRIT fut pour moi un réel enrichissement.

J'ai également eu l'opportunité d'établir une collaboration avec Rodrigo Neves Calheiros, Post-Doctorant du *CLOUDS Laboratory* de Melbourne dirigé par Prof. Rajkumar Buyya, que je remercie pour les travaux que nous avons pu mener ensemble.

Un grand merci à Khalil Drira, notre responsable d'équipe au LAAS, avec qui j'ai organisé les journées du thème RC, enrichissantes à de nombreux points de vue et appréciées de tous. Merci à Jean-Marc Pierson de m'avoir accueilli au sein de l'équipe SEPIA de l'IRIT dont il est le responsable.

Amical remerciement à Sonia De Sousa, secrétaire de notre équipe SARA du LAAS, qui m'a aidé à l'organisation des journées du thème RC et largement facilité de nombreuses démarches.

J'adresse des sincères remerciements aux directeurs de mes deux laboratoires : Jean Arlat, directeur du LAAS, et Michel Daydé, directeur de l'IRIT, qui ont largement soutenu ma candidature pour un futur Post-Doctorat.

Chaleureux remerciements à l'ensemble de mes amis, déjà docteurs ou en voie de le devenir, ainsi qu'à toutes les personnes que j'ai pu côtoyer dans mes deux laboratoires. Mes *collègues* et non moins amis de bureau : Maia qui a quotidiennement égayé nos journées avec son humour sans égal..., Ane qui a également amené avec elle la bonne humeur du Pays Basque dans notre bureau, et enfin, Samir, qui nous a rejoint à l'entame de ma dernière année de thèse et qui m'a été d'une aide précieuse lors de mes derniers travaux, aussi en partageant ses larges connaissances lors de nombreuses discussions dans divers domaines. Merci à mes amis : Ahmad Al Sheikh, Aurélien Gonzalez, Jean-Marie Codol et Rémi Sharrock ansi que Andreea Simona Anghel et Bogdan Prisacari auprès de qui j'ai découvert le monde de la recherche depuis mon stage de Master 2. La famille Ben Alaya, Ghada et Mahdi, et les amis du bureau A45 que j'ai souvent empêché de travailler afin de passer des fins de journées agréables : Tanya, Henda, Josu et Josselin, ainsi que tous les autres doctorants ou post-doctorants de notre équipe SARA et de l'équipe SEPIA de l'IRIT.

Enfin, joyeux remerciements à mes précieux parents, Annie et Michel, qui me soutiennent continuellement et bien évidemment à mon irremplaçable grande sœur, Laure.

L'ignorance a le mépris facile.

Ce qui se conçoit bien s'énonce clairement, et les mots pour le dire arrivent aisément. (Boileau)

L'ignorance est le plus grand des mépris.

Résumé

Ces dernières années, de nouvelles problématiques sont nées au vu des considérations écologiques de plus en plus présentes dans notre société. Dans le domaine de la technologie de l'Information, les centres de calcul consomment actuellement environ 1.5% de l'électricité mondiale. Cela ne cesse d'augmenter en raison de l'évolution de nombreux domaines et particulièrement du *Cloud Computing*. Outre cet aspect environnemental, le contrôle de la consommation d'énergie fait désormais partie intégrante des paramètres de Qualité de Service (QoS) incombant aux fournisseurs de services de *Cloud Computing*. En effet, ces fournisseurs de services à la demande proposent à leurs utilisateurs un contrat de QoS, appelé SLA (*Service Level Agreement*), qui définit de manière précise la qualité de service qu'ils s'engagent à respecter. Le niveau de QoS proposé influence directement la qualité d'utilisation des services par les utilisateurs, mais aussi la consommation et le rendement général de l'ensemble des ressources de calcul utilisées, impactant fortement les bénéfices des fournisseurs de services.

Le *Cloud Computing* étant intrinsèquement lié à la virtualisation des ressources de calcul, une élaboration de modèles d'architecture matérielle et logicielle est proposée afin de définir les caractéristiques de l'environnement considéré. Ensuite, une modélisation détaillée de paramètres de QoS en termes de performance, de sûreté de fonctionnement, de sécurité des données et de coûts est proposée. Des métriques mesurables et réutilisables associées à ces paramètres sont définies afin d'étendre les possibilités de mesures et d'évaluation des SLA. Ces modélisations constituent la première contribution de cette thèse.

Il convient alors de démontrer comment l'utilisation et l'interprétation de plusieurs métriques de QoS ouvrent la possibilité d'une analyse plus complexe et plus fine de la perspicacité des algorithmes de placement. Celles-ci pouvant être alors utilisées par les fournisseurs de service afin de configurer leur système. Cette approche multi-critères leur apporte des informations importantes sur l'état de leur système qu'ils peuvent analyser afin de gérer le niveau de chaque paramètre de QoS. Ainsi, quatre métriques antagonistes, incluant la consommation énergétique, ont été sélectionnées et utilisées conjointement dans plusieurs algorithmes de placement de manière à montrer leur pertinence, l'enrichissement qu'elles apportent à ces algorithmes, et comment un fournisseur de service peut tirer profit des résultats d'une optimisation multi-objectifs. Cette seconde contribution présente un algorithme génétique (GA) ainsi que deux algorithmes gloutons. L'analyse du comportement de l'algorithme génétique a permis de démontrer différents intérêts

d'une optimisation multi-critères appliquée à des métriques de QoS habituellement ignorées dans les études dédiées au *Cloud Computing*.

La troisième contribution de cette thèse propose une étude de l'impact de l'utilisation des métriques de QoS sur l'ordonnancement de machines virtuelles au cours du temps. Pour cela, le simulateur CloudSim a été exploité et étendu afin d'améliorer ses fonctionnalités de gestion de consommation énergétique. Tout d'abord par l'ajout du DVFS (*Dynamic Voltage & Frequency Scaling*) apportant une gestion dynamique très précise des fréquences de fonctionnement CPU, puis la possibilité de reconfiguration de machines virtuelles et enfin par la gestion dynamique des évènements. Les simulations effectuées mettent en jeu l'ensemble de ces outils énergétiques ainsi que les algorithmes de placement et évaluent chacune des métriques de QoS sélectionnées. Ces simulations donnent une vision temporelle de l'évolution de celles-ci, en fonction des algorithmes utilisés et de plusieurs configurations d'optimisation de l'algorithme génétique. Cela permet d'analyser sous différents angles le comportement des algorithmes gloutons, l'impact des optimisations du GA, et l'influence des métriques les unes par rapport aux autres. Enfin, durant cette thèse, une collaboration à l'international a pu être établie avec le laboratoire australien *Cloud Computing and Distributed Systems (CLOUDS) Laborartory* de Melbourne, dirigé par Prof. Rajkumar Buyya grâce aux différentes évolutions apportées au simulateur CloudSim.

Publications scientifiques

- **Articles de revues internationales**
  - Tom Guérout, Samir Medjiah, Georges Da Costa, Thierry Monteil. **Quality of Service Modeling for Green Scheduling in Clouds.** Dans : *Sustainable Computing*, Elsevier, août 2014. Accès : http ://www.sciencedirect.com/science/article/pii/S221053791400047X

  - Tom Guérout, Georges Da Costa, Thierry Monteil, Rodrigo Neves Calheiros, Rajkumar Buyya, Mihai Alexandru. **Energy-aware simulation with DVFS.** Dans : *Simulation Modelling Practice and Theory*, Elsevier, Numéro spécial Energy efficiency in Grids and Clouds, Vol. 39, p. 76-91, 2013. Accès : http ://www.sciencedirect.com/science/article/pii/S1569190X13000786

- **Conférences et workshops internationaux**
  - Tom Guérout, Mahdi Ben Alaya. **Autonimic energy-aware task scheduling** (regular paper). Dans : *IEEE International Conference on Collaboration Technologies and Infrastructures (WETICE 2013)*, Hammemet, Tunisie, 17/06/2013-20/06/2013, IEEE Computer Society - Conference Publishing Services, (en ligne), juin 2013.
    Accès : http://www.laas.fr/pulman/pulman-isens/web/app.php/publiPerso/1000010284

  - Mahdi Ben Alaya, Thierry Monteil, Khalil Drira, Tom Guérout. **A framework to create multi-domains autonomic middleware** (regular paper). Dans : *International Conference on Autonomic and Autonomous Systems (ICAS 2012)*, St Marteen (Pays Bas), 25/03/2012-30/03/2012, LAAS, (en ligne), mars 2012.
    Accès : http ://www.laas.fr/pulman/pulman-isens/web/app.php/publiPerso/1000010284

  - Remi Sharrock, Patricia Stolf, Thierry Monteil, Tom Guérout. **Internal self-protecting for consistency and stability in an autonomic manager** (regular paper). Dans : *IEEE International Symposium on Network Computing and Applications (IEEE NCA 2011)*, Toulouse, 21/10/2011-23/10/2011, IEEE Computer Society, ISBN : 978-0-7695-4550-9, p. 44-49, 2011.

- **Conférences et workshops nationaux**
  - Tom Guérout, Thierry Monteil, Georges Da Costa, Mihai Alexandru. **Simulation énergétique de tâches distribuées avec changements dynamiques de fréquence** (short paper). Dans : *Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS 2013)*, INRIA Grenoble, 15/01/2013-18/01/2013, LAAS, (en ligne), janvier 2013.
    Accès : http ://www.laas.fr/pulman/pulman-isens/web/app.php/publiPerso/1000010284

  - Tom Guérout, Thierry Monteil, Georges Da Costa, Mihai Alexandru. **Grid'5000 energy-aware experiments with DVFS** (short paper). Dans : *Grid'5000 school 2012*, Nantes, 03/12/2012-06/12/2012, LAAS, (en ligne), décembre 2012.
    Accès : http ://www.laas.fr/pulman/pulman-isens/web/app.php/publiPerso/1000010284

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Sommaire

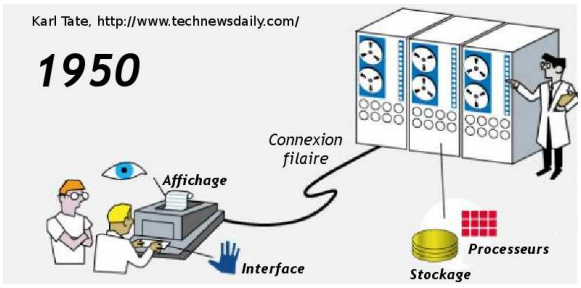
En cette année 2014, voici plus de 65 ans que l'un des tout premiers ordinateurs électroniques fut mis en service, dénommé EDVAC (*Electronic Discrete Variable Automatic Computer*). Le *30 juin 1945*, John Von Neumann publie la première version d'un rapport [1] sur l'EDVAC contenant la première discussion documentée sur la notion de programme enregistré et le premier plan de l'architecture d'un ordinateur. Sa mise en service sera effective qu'en 1951 en raison d'un conflit sur le brevet entre l'*University of Pennsylvania* et le duo Eckert & Mauchly. L'ère des ordinateurs électroniques est alors en route et montre déjà des capacités mettant à mal les anciens systèmes électromécaniques : 3 secondes contre 15 minutes nécessaires au calcul d'une table de tir. Malgré cette avancée importante, les premiers ordinateurs des années 50', appelés *Mainframe* ou "ordinateurs centraux", étaient physiquement très imposants : une surface de 45.5 m^2 pour 7850 Kg pour l'EDVAC. Celui-ci nécessitait également une équipe de près de 100 personnes pour assurer son fonctionnement. Cela représentait donc un système centralisé sur lequel une connexion filaire permettait à chaque utilisateur d'accéder aux mêmes données centralisées et d'utiliser les mêmes unités de calcul, comme représenté sur la Figure 1.1a.

L'amélioration des procédés de fabrication de composants électroniques ont amené, à partir des années 60, la fabrication d'ordinateurs à la fois bien moins encombrants et possédants des capacités de calcul nettement supérieures communément appelés "micro-ordinateur". C'est ainsi que le premier micro-ordinateur, "Micral N", fut breveté par le français François Gernelle en 1973. Cette miniaturisation des composants

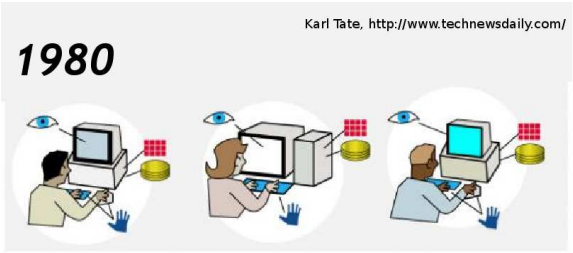
1. Copie du rapport de John Von Neumann (1945) :
http ://www.virtualtravelog.net/wp/wp-content/media/2003-08-TheFirstDraft.pdf

électroniques a donc permis une utilisation personnelle des ordinateurs, offrant à chaque utilisateur une capacité de stockage et de calcul dédiée. Lors de cette même décennie, l'étude de la mise en réseau de ces terminaux à également permis une avancée spectaculaire.

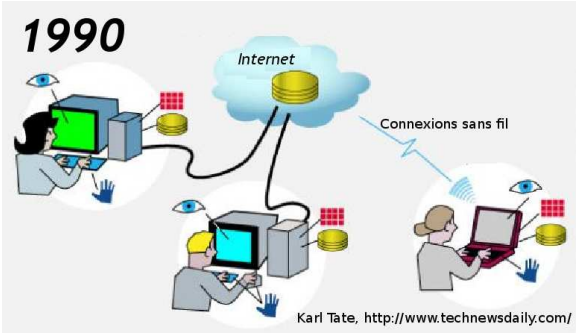
C'est en 1968 que le premier ordinateur spécialisé, appelé IMP (*Interface Message Processor*), permettant l'inter-connexion de plusieurs autres ordinateurs est apparu. Les notions de commutation de paquets et de protocoles de communication assurant l'envoi et la réception des paquets furent introduites, autorisant ainsi la circulation d'informations d'un ordinateur à un autre via l'IMP. Cette technique de mise en réseau, illustrée par la Figure 1.1b, permet aux utilisateurs de partager et d'échanger leurs données, toujours stockées localement sur chacun de leur ordinateur. Cette évolution amenant à la mise en réseau local des ordinateurs ne cessa de se développer et fut une évolution marquante pour l'ère de l'Information (IT). La mise en réseau prit de l'ampleur dans les années 90' avec l'arrivée d'Internet (contraction anglaise de *International Network*) et déjà introduit en 1972 par Robert Kahn [2]. Cette nouvelle évolution permet aux utilisateurs le téléchargement de contenus stockés sur des machines distantes, comme illustrée en Figure 1.1c.



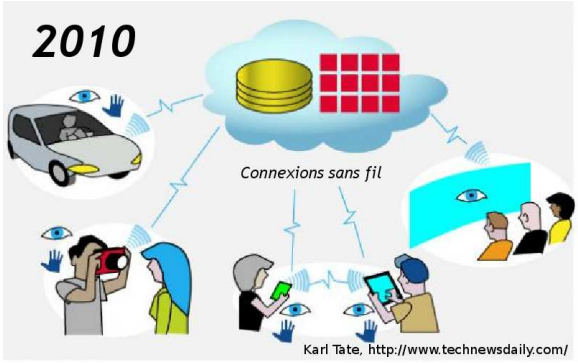
(a) Ordinateur central : Une unité de calcul pour une centaine de personnes



(b) Ordinateurs personnels et mise en réseau : un ordinateur par personne, début de partage d'informations locales



(c) Apparition de l'Internet : mise en réseau à l'échelle mondial, début du téléchargement d'informations



(d) Nouvelle ère : Internet des objets et *Cloud Computing*. Ressources délocalisées et partagées par des milliers d'utilisateurs

Figure 1.1 – Évolution des systèmes informatiques et de leurs utilisations

2. Robert Kahn : http ://fr.wikipedia.org/wiki/Robert_E._Kahn

Après plusieurs étapes d'évolution de l'Internet, l'ère de l'Information d'aujourd'hui voit une nouvelle manière d'accéder et d'utiliser des ressources distantes. De plus, ce ne sont plus uniquement les ordinateurs personnels qui sont capables d'utiliser ces ressources, mais toutes sortes d'objets connectés, illustrés par la Figure 1.1d. Ces ressources distantes ont pour but de rendre transparent le traitement de l'information aux yeux de l'utilisateur et ont également la propriété d'être accessibles indépendamment de la géo-localisation de l'utilisateur. Ce concept d'utilisation de ressources distantes via n'importe quel type de dispositif et indépendant du lieu d'utilisation a été introduit sous le nom de *Ubiquitus Computing*, terme utilisé pour la première fois par Mark Weiser en 1988, dans lequel s'inscrivent les concepts de Systèmes Cyber-Physique et l'Internet des Objets. Cette nouvelle ère de l'information a également vu naître un nouveau paradigme, dédié à la mise en place de services utilisant ces ressources distantes, le *Cloud Computing* [8, 115, 94, 49, 152]. Le concept de services de *Cloud Computing* réside dans le fait de proposer d'utiliser ces services distants, gérés par une infrastructure externe, plutôt que d'installer et utiliser une application localement sur un ordinateur personnel.

Les centres de calcul hébergeant les services de *Cloud Computing*, composés désormais de plusieurs milliers de machines rassemblées dans un *data center* et séparées en centaines de *clusters*, voient leur utilisation grandissante, due aux performances des ressources, aux services et à la transparence d'utilisation et de fonctionnement qu'ils proposent [127]. En effet, ce nouveau paradigme est apparu au milieu des années 2000, largement aidé par l'évolution des *data centers* de Amazon[3] qui, comme la plupart des réseaux informatiques, utilisait couramment seulement environ 10% de la capacité de leurs *data centers*, pour pouvoir couvrir des pics occasionnels d'utilisation.

C'est en 2006, que Amazon se lance dans un développement plus avancé de leurs *data centers* afin de faire émerger la première plate-forme de services de type *Cloud Computing*, apparue sous le nom de AWS (*Amazon Web Services*). Par la suite, quelques grands noms des fournisseurs de *Cloud Computing* d'aujourd'hui font leur apparition tels que : Saleforce[4] en 2007, Google App Engine [153, 56] en 2008, Microsoft Azure[5] en 2010 ou encore Oracle.

À ce jour, le concept même du *Cloud Computing* n'obtient pas l'aval de tout le monde. Des grands noms de l'informatique d'aujourd'hui voient dans le *Cloud Computing* une manière de mettre en avant un nouvel aspect de l'informatique correspondant totalement à un phénomène de mode, présenté comme facilitant l'utilisation d'applications par des utilisateurs lambda, et repoussant encore un peu plus la dépendance de ces utilisateurs à des services connectés accessibles et utilisables à tous moments. D'autres aspects, plus techniques, sont remis en question et peuvent pousser à se demander si le *Cloud Computing* ne posera pas plus de soucis que ce qu'il pourra apporter comme solutions innovantes pour l'informatique, notamment à cause de l'externalisation des données qui semble être le sujet le plus sensible quant à la sécurité des services proposés.

Passés ces points de discordes sur les fondements du *Cloud Computing*, ce nouveau paradigme attire sans aucun doute un grand nombre d'utilisateurs, entreprises ou particuliers, grâce entre autres aux principes de services à la demande et aux coûts qui sont fonction de leurs utilisations (principe du *pay-as-you-go*).

3. Amazon's early efforts in cloud computing partly accidental : http ://itknowledgeexchange.techtarget.com/cloud-computing/2010/06/17/amazons-early-efforts-at-cloud-computing-partly-accidental/
4. http ://www.salesforce.com/fr/
5. http ://www.azure.microsoft.com/

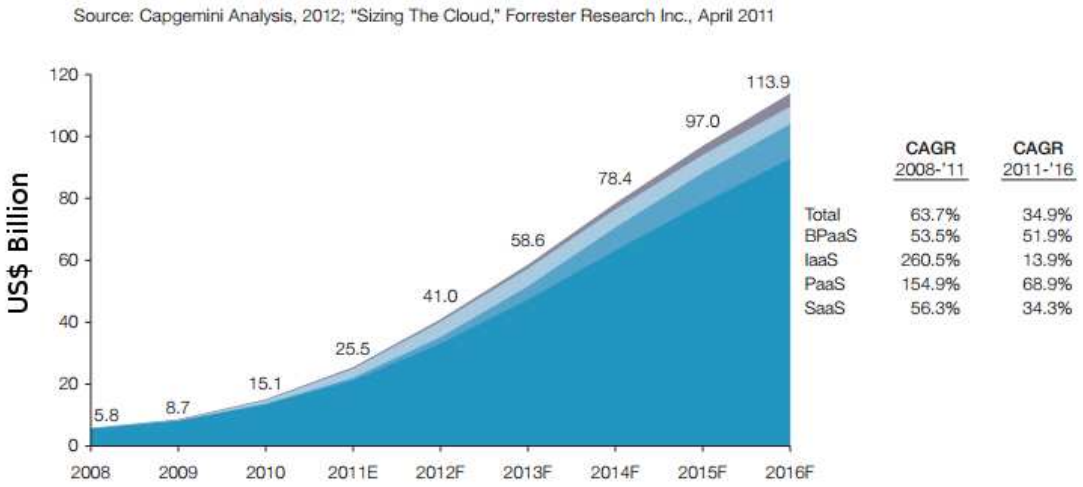


Figure 1.2 – Aperçu et estimation de l'évolution des revenus du marché des *Clouds* publics, de 2008 à 2016 (prévisions), en Milliard de dollar. L'abréviation (CAGR) *Compound Annual Growth Rate* représente le taux de croissance annuelle.

Afin d'illustrer la montée exponentielle du marché du *Cloud Computing* dans le monde, la Figure 1.2 (tirée d'une étude de 2011 de CapGemini [6]) illustre l'évolution de la masse budgétaire créée par le *Cloud Computing* ainsi qu'une estimation de son évolution jusqu'à l'horizon 2016. La liste des fournisseurs de services de *Cloud Computing* s'étend désormais à plus de 900 propositions [7]. Au vu de cette montée en puissance du *Cloud Computing* il est indiscutable que ce nouveau paradigme est actuellement au sein d'une spirale ascendante et que son incroyable développement secoue le monde de l'informatique.

Du point de vue de l'utilisateur, le *Cloud Computing* séduit grâce à ses caractéristiques :

- Service à la demande (*self-service*)
- Coût fonction de l'utilisation (*pay-as-you-go*)
- Rapidité
- Accès permanent, etc...

Ces fonctionnalités sont les fondations de l'immense capacité que constitue un environnement de *Cloud Computing* en termes de puissance de calcul et de modèle économique de location de services. Pour assurer les différentes caractéristiques citées ci-dessus, un fournisseur de services de *Cloud Computing* doit faire face à de nombreuses problématiques techniques. Que ce soit par rapport à la performance, à la mise en place de fonctionnalités de sécurisation des services ou à l'aspect purement budgétaire et économique qui est la première raison de survie d'un fournisseur de services de *Cloud Computing*.

6. Document CapGemini : http ://www.au.capgemini.com/resource-file-access/resource/pdf/Cloud_Computing_in_the_Property__Casualty_Insurance_Industry.pdf
7. http ://http ://www.spamina.com/eng/cloud_hosting_providers_list.php

Plusieurs définitions du concept de *Cloud Computing* ont été proposées [138, 155, 27]. Selon NIST [8] (*National Institute of Standards and Technology*), le *Cloud Computing* peut être défini comme un modèle permettant l'accès à un ensemble de ressources configurables aux niveaux : réseaux, processeurs, mémoire et stockage, qui peuvent être aussi bien mises à disposition puis libérées très rapidement par les utilisateurs, sans poser de gros soucis de gestion du côté du fournisseur de ressources.

Ces accès aux ressources peuvent se faire à différents niveaux, offrant plus ou moins d'outils et de possibilités de gestion à l'utilisateur. Les trois principaux niveaux de services de *Cloud Computing* proposés par les fournisseurs sont les suivants :

Définition 1.1 : Software as a Service (SaaS)

Dans ce modèle SaaS de location de service, le consommateur utilise une application *Cloud*, mais il ne contrôle pas le système d'exploitation, le matériel ni les périphériques réseau de l'environnement d'approvisionnement de l'application. Les applications peuvent concerner différents domaines : ERP (*Enterprise Resource Planning*), CRM (*Customer Relationship Management*), Analytique, logistique, applications métier, etc. Fournisseurs actuels : SalesForce (CRM à la demande) et Google Apps, Antenna Software [9], Cloud9 Analytics [32], CVM Solutions [10] entre autres.

Définition 1.2 : Platform as a Service (PaaS)

Au niveau PaaS, un environnement d'hébergement est mis à la disposition des consommateurs pour leurs applications. Il a la capacité de contrôler les applications dans l'environnement hôte. Toutefois, le consommateur dispose d'un contrôle limité du système d'exploitation, du matériel et des périphériques réseau de l'environnement d'hébergement. Fournisseurs actuels : Google App Engine [153], Engine Yard [11], Red Hat OpenShift [12], Heroku [13].

Définition 1.3 : Infrastructure as a Service (IaaS)

Dans ce modèle IaaS, le consommateur dispose d'un accès complet à toutes les ressources : processeur, stockage, mémoire, périphériques réseau. Le consommateur a également la possibilité de contrôler le système d'exploitation, le matériel, l'infrastructure réseau et les applications déployées. Fournisseurs actuels : Amazon AWS [6], Google Compute Engine [57], Microsoft Azure [97], IBM SmartCloud Enterprise [65], Rackspace Cloud [111], HP Enterprise Converged Infrastructure [64].

D'autres catégories de services sont également connues :

Définition 1.4 : Network as a Service (NaaS)

8. http ://www.nist.gov/itl/csd/cloud-102511.cfm
9. http ://www.antennasoftware.com/
10. http ://www.cvmsolutions.com/
11. https ://www.engineyard.com/
12. https ://www.openshift.com/
13. https ://www.heroku.com/

Il permet la création de réseau à la volée entre machines virtuelles et de gérer sa configuration. Ces services sont très souvent associés avec des SDN (*Sofware Design Network*) qui facilitent énormément la manière de configurer et de gérer son réseau. Exemple : OpenNaas [14]

Définition 1.5 : Data as a Service (DaaS)

Le DaaS permet l'utilisation de données délocalisées. Ce type de service est utilisé par des applications composites, dites *mashup* dont le rôle est de corréler des données venant de milieux hétérogènes. Exemples : Factual [15] et InfoChimps [16].

Définition 1.6 : STorage as a Service (STaaS)

Cette catégorie de service offre des solutions de stockage de fichiers chez des prestataires externes tels que : Amazon S3 [7], iCloud [17], SkyDrive [18], Wuala [19], Google Drive, Ubuntu One ou Dropbox.

Définition 1.7 : Business Process as a service (BPaaS)

Il fournit des services regroupant des applications qui intègrent la logique, les données et les processus de *business* des entreprises.

Définition 1.8 : Desktop as a Service (DaaS)

Le DaaS est présenté comme une solution dématérialisée qui permet, tout comme le *Virtual Desktop Infrastructure* (VDI), de totalement dissocier la machine traitant l'affichage, de la machine de l'utilisateur.

Différentes approches d'utilisation (ou modèles de déploiement [92, 124, 149, 154]) des services *Cloud* sont définies :

Définition 1.9 : Cloud Public

Un *Cloud* public peut être décrit comme un moyen de rendre des services accessibles à tous les utilisateurs d'Internet. Le terme "public" ne signifie pas nécessairement que les services sont gratuits, mais ils peuvent être assez peu coûteux à utiliser. Cela ne signifie pas non plus que les données des utilisateurs sont rendues publiques et visibles par les autres utilisateurs. Les fournisseurs de *Cloud* public s'efforcent de mettre en place des politiques de sécurité, tout en conservant des services performants à coûts réduits.

Définition 1.10 : Cloud Privé

Ce modèle de déploiement est utilisé par les organisations pour une utilisation interne restreinte à celles-ci. Il offre les mêmes fonctionnalités qu'un *Cloud* public, telle que l'élasticité, à la seule différence

14. http ://opennaas.org/
15. http ://www.factual.com/
16. http ://www.infochimps.com/
17. https ://www.icloud.com/
18. https ://onedrive.live.com
19. https ://www.wuala.com/fr/

que son accès est contrôlé et réservé aux membres de l'entreprise. Les utilisateurs d'un *Cloud* privé ont une marge de manœuvre plus large sur le contrôle de l'infrastructure sans aucune restriction de performance ni de sécurité. En outre, son utilisation étant réservée aux employés de l'entreprise possédant le *Cloud*, tout type de problème juridique est écarté.

Définition 1.11 : Cloud Hybride

Un *Cloud* hybride est un environnement de *Cloud Computing* dans lequel une organisation fournit et gère des ressources internes et externes. Par exemple, une organisation peut utiliser un service de *Cloud* public, tel que *Amazon Simple Storage (Amazon S3)* pour archiver des données, mais continuer à maintenir le stockage interne pour les données des clients opérationnels. L'approche hybride permet à une entreprise de tirer parti des performances et des faibles coûts qu'un environnement de *Cloud Computing* public offre, sans exposer les applications et les données critiques au sein de la partie publique du *Cloud* utilisé. L'utilisation conjointe de *Cloud* public et privé nécessite une stratégie de déploiement tenant compte de la configuration, du contrôle des changements, de la sécurité, de la gestion des pannes et de la budgétisation afin d'être la plus efficace possible. Ce type de *Cloud* est aussi appelé *hybride IT*.

Définition 1.12 : Community Cloud

Les *Clouds* communautaires sont le résultat d'une collaboration entre plusieurs entreprises souhaitant partager leurs infrastructures. Ces regroupements se font entre des entreprises ayant des intérêts communs (sécurité, conformité, juridiction, etc) et peuvent être gérés et/ou hébergés en interne ou par une tierce entreprise. Bien sûr, les coûts liés à l'utilisation de ce type de *Cloud* sont assez importants, de la même manière que les *Clouds* privés, dû au faible nombre d'utilisateurs.

Les quelques points d'histoire de l'évolution de l'informatique et des principales caractéristiques du *Cloud Compuing* exposés ci-dessus permettent d'analyser le changement du mode de gestion des machines par les hommes mais aussi l'impact que les machines peuvent désormais avoir sur les hommes. Le *Cloud Computing* est donc l'une des dernières évolutions de l'ère de l'Information, regroupant un nombre de machines bien plus important que le nombre de personnes ayant la lourde charge de les faire fonctionner. Le nombre de machines mises en jeu, le bon fonctionnement des services proposés et les considérations financières des fournisseurs de services rendent la gestion de ce nouveau paradigme très complexe dans de nombreux domaines.

Désormais, le nombre grandissant de propositions de services de *Cloud Computing* entraîne également de nombreuses études de recherche dans toutes les problématiques que ce nouveau paradigme soulève, notamment en matière de QoS (*Quality Of Service* en anglais). La QoS regroupe tous les paramètres d'une infrastructure de fournisseurs de services *Cloud* et relie fortement celui-ci aux utilisateurs. La QoS dédiée au *Cloud Computing* inclut l'ensemble des paramètres auxquels un fournisseur de services se doit de porter attention, balayant divers domaines, de la performance pure de son système jusqu'à l'impact environnemental qui lui incombe en passant par tous les aspects de coûts, de performance et de sécurité de fonctionnement. Il convient aussi de rappeler l'importance de l'aspect financier pour un fournisseur de services, et l'analyse des tous ces paramètres de QoS doit pouvoir lui permettre de trouver le meilleur

compromis entre le coût de fonctionnement de son *data center*, la qualité des services qu'il propose et le bénéfice qu'il peut en tirer.

La section suivante présente les domaines d'étude de cette thèse liés au *Cloud Computing* en introduisant les nombreuses problématiques qu'ils soulèvent. Celles-ci constituent les bases de l'ensemble des travaux effectués, exposés dans ce manuscrit.

1.1 Cloud Computing : Qualité de Service et enjeux énergétiques

Du point de vue du fournisseur de services, la vision du domaine est d'un côté fortement liée à celle des utilisateurs, avec comme enjeu de satisfaire leurs attentes, et est à la fois tout autre avec le besoin de pouvoir évaluer la qualité et le rendement de leur système afin d'en assurer la rentabilité. Ces deux considérations amènent les fournisseurs de services de *Cloud Computing* à analyser bien des caractéristiques de leurs systèmes afin de les améliorer constamment pour qu'ils intègrent de nouvelles technologies profitant autant aux utilisateurs qu'à leurs propres intérêts. En effet, l'aspect économique du *Cloud Computing* n'entraîne pas seulement l'analyse financière du coût de location des services, mais également l'étude de nombreux paramètres de qualité de service, qui chacun ont leur influence sur le fonctionnement global du système.

L'utilisation de services de *Cloud Computing* est basée sur la définition d'un contrat, appelé SLA (*Service Level Agreement*), qui lie le fournisseur des services à ses utilisateurs. Ainsi, dans un SLA se trouve un ensemble de clauses, que le fournisseur s'engage à respecter. Celle-ci peuvent concerner divers domaines :

- Performance du matériel
- Disponibilité des services
- Sécurisation des données
- Coûts de location des machines virtuelles, etc...

Un SLA contient un ensemble de paramètres dont le fournisseur de services doit tenir compte. Ce dernier doit s'efforcer de maintenir un niveau de prestation, afin tout d'abord de respecter les termes du contrat SLA qu'il promet aux utilisateurs. Le non respect de ces engagements pourrait signifier que sa plate-forme n'est pas gérée de façon à pouvoir satisfaire les besoins des utilisateurs, mais surtout son incapacité à respecter le niveau de service auquel il s'est engagé dans le contrat de SLA. Ainsi, sa crédibilité aux yeux des utilisateurs est largement mise en jeu lors de la définition de ces contrats. Outre ces paramètres de QoS influençant directement la qualité d'utilisation des services, un fournisseur a aussi pour principal but de tirer profit des locations de ses services, et s'assurer du bon fonctionnement et du bon rendement de l'ensemble de sa plate-forme. C'est aussi en cela que l'analyse de paramètres de qualité de service joue un rôle important pour le fournisseur de services, poussant ainsi celui-ci à faire des choix cruciaux quant à la configuration de son système.

En effet, les paramètres de qualité de service concernant le *Cloud Computing* peuvent être classifiés dans des catégories bien distinctes (performance, coûts, ...) et donc être représentés par des propriétés bien différentes (temps de réponse, consommation énergétique, extensibilité, intégrité, ...). Cet ensemble de paramètres met donc en jeu des aspects de qualité de service très hétérogènes. Ces paramètres de qualité

de service concernant le *Cloud Computing* ne vont pas tous dans la même direction, et peuvent ainsi poser des problèmes de compromis très complexes à résoudre pour les fournisseurs de services. Cette difficulté réside dans le fait de trouver un compromis pouvant satisfaire à la fois le fournisseur de services et ses utilisateurs. Le rôle du fournisseur de services est donc de prendre conscience de ces différents paramètres influençant le fonctionnement de son système et de trouver une solution de configuration pour celui-ci qu'il estime être la meilleure pour une situation donnée. Les compromis de qualité de service sont très souvent étudiés comme une relation conflictuelle entre la performance d'un système (temps de réponse, rapidité d'exécution) et la puissance nécessaire à celui-ci pour garantir cette performance. Cet exemple est un cas typique de paramètres antagonistes donnant lieu à un compromis entre les deux. Dans le cadre du *Cloud Computing* les résultats de ces compromis sont en relation directe avec les termes des SLA proposés par le fournisseur de services. Connaissant le niveau de qualité de service qu'il s'est engagé à fournir à ses utilisateurs, il se doit de prendre une décision ne détériorant pas de façon excessive les paramètres inclus dans les SLA tout en faisant attention à garder un rendement (financier et/ou énergétique) favorable. L'exemple présenté ci-dessus ne met en jeu que deux paramètres de qualité de service et constitue déjà un problème complexe à résoudre. De nos jours, la prise en compte unique de ces deux paramètres ne représente plus la réalité des caractéristiques de qualité de service rentrant en jeu dans les SLA. En effet, au sein des SLA se trouvent des règles concernant bien des paramètres différents (autres que le temps de réponse). Il est donc très important pour le fournisseur de services de pouvoir analyser l'ensemble (dans le meilleur des cas) des caractéristiques mises en jeu dans ces SLA afin d'avoir un aperçu global de l'état de son infrastructure en termes de QoS. Cela dans le but d'apporter à ses utilisateurs la meilleure qualité de service possible, mais aussi d'éviter les risques de violations d'une ou plusieurs règles de SLA. Prendre en compte plusieurs paramètres de qualité de service, exprimant une propriété réelle de l'infrastructure et ayant un impact à la fois sur les conditions d'utilisation des utilisateurs et sur le rendement de celle-ci est donc très intéressant.

Une telle approche complexifie donc d'autant plus les prises de décisions quant à la configuration du système. Plus le nombre de paramètres étudiés est grand plus un compromis satisfaisant est difficile à trouver. De plus, le rendement financier d'un *data center* de *Cloud Computing* est très largement influencé par sa consommation énergétique. Un fournisseur de services se retrouve donc constamment dans une situation dans laquelle il doit gérer le rendement énergétique global de ses ressources, tout en gardant en tête les enjeux de qualité de service qu'il a promis à ses utilisateurs. En effet, les problèmes de consommation d'énergie dû à l'évolution de l'industrie mondiale de ces cinquante dernières années, a soulevé de nombreux débats quant à l'impact environnemental que celle-ci entraîne. Les technologies de l'Information (IT), et notamment les *Clouds* n'en sont pas privés, suite à plusieurs études démontrant que les consommations énergétiques de l'IT sont croissantes et donc de plus en plus problématiques pour la société actuelle.

Le rapport de Jonathan G. Koomey [81], intitulé "*Growth in data center electricity use 2005 to 2010*", bien que datant de l'année 2010, fait état de l'évolution de l'utilisation et de la consommation énergétique des centres de calcul, de plus en plus élevée. En effet, le 2 Août 2007, en réponse à la loi publique 109-431 [20] le programme *EPA ENERGY STAR* publie un rapport [21] évaluant les possibilités d'améliorations

20. https ://www.energystar.gov/ia/products/downloads/Public_Law109-431.pdf
21. http ://hightech.lbl.gov/documents/data_centers/epa-datacenters.pdf

de l'efficacité énergétique des serveurs informatiques gouvernementaux et commerciaux et *data centers* aux États-Unis.

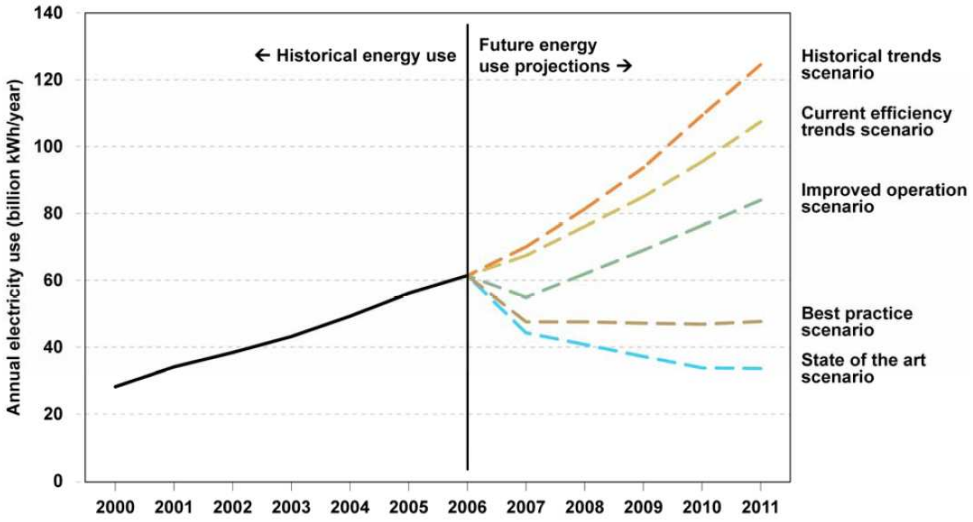


Figure 1.3 – Évolution et prévision de la consommation d'énergie des équipements informatique aux État-Unis en 2006.

Les indications contenues en Figure 1.3 proviennent d'une étude datant de 2006. Bien que celle-ci ait maintenant déjà plus de 8 ans, une telle analyse de prévisions de la consommation d'énergie des équipements informatiques reste tout à fait d'actualité. Cela, dû au fait du taux de croissance de l'utilisation de l'ensemble des systèmes de l'IT, et notamment du *Cloud Computing* boosté par la variété toujours plus vaste des services proposés.

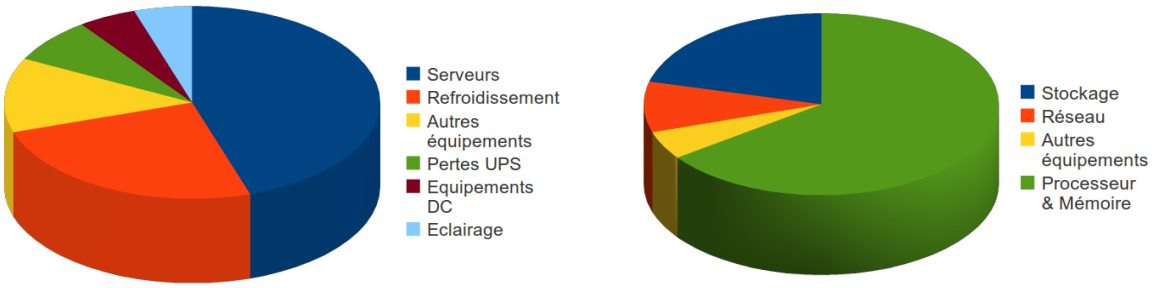
Les différentes courbes de la Figure 1.3 correspondent à différents scénarios envisagés. L'application parfaite des scénarios sur l'état de l'art impliquent des changements plus que significatifs dans les *data centers* qu'il serait possible de mettre en œuvre uniquement lors de rénovations très importantes des installations. Il a été supposé dans ces scénarios que les investissements financiers supplémentaires nécessaires aux rénovations de ces infrastructures soient appliqués seulement sur 50% du parc des *data centers*. Aussi, pour l'ensemble du matériel informatique utilisé, il a été supposé que la totalité de ces équipements serait remplacée sur une période de cinq ans.

Ces scénarios, sur la base des hypothèses décrites ci-dessus, illustrent un potentiel important pour les technologies et les pratiques efficaces pour améliorer l'efficacité énergétique des serveurs et des *data centers* entre 2006 et 2011 :

- Le scénario *state-of-the-art* pourrait réduire la consommation d'électricité jusqu'à 55% par rapport aux tendances de 2006 en matière d'efficacité, ce qui représente le potentiel technique maximal.
- Le meilleur scénario de pratique pourrait réduire la consommation d'électricité jusqu'à 45% par rapport aux tendances de 2006, avec des gains d'efficacité qui pourraient être réalisés en utilisant les technologies d'aujourd'hui.

- Le scénario de gestion opérationnelle améliorée offre des économies d'électricité de plus de 20% par rapport aux tendances de 2006, ce qui représente des possibilités d'économies d'énergie à faibles coûts.

Une autre étude plus récente[22] (2012), fait état de l'évolution de la consommation énergétique uniquement due au *Cloud Computing*. En 2007, cette consommation était évaluée à 623 Millions de kWh ce qui plaçait le *Cloud Computing*, en termes de consommation énergétique, au-dessus de celle de pays comme le Brésil, la France, le Canada ou l'Allemagne. Au vu de l'évolution constatée de nos jours, cette consommation est estimée à plus du triple de ce qu'elle était en 2007, soit environ 2000 Millions de kWh, ce qui équivaut à la somme des consommations de ces quatre pays.



(a) Répartition de la consommation d'énergie entre tous les équipements dans un *data center*.

(b) Répartition de la consommation d'énergie entre les composants d'un serveur.

Figure 1.4 – Diagrammes circulaires de la répartition de la consommation d'énergie.

Cette prise de conscience se répercute évidemment dans le domaine de l'IT, et incite les fournisseurs de services de *Cloud Computing* à rendre leurs équipements de moins en moins gourmands en énergie. De plus, cela les amène également à intégrer des techniques de gestion de leurs systèmes afin d'utiliser le moins de ressources possibles et ainsi diminuer la consommation énergétique de l'ensemble de leurs dispositifs. Si l'on s'intéresse à la répartition des consommations d'énergie par composant physique d'une machine, illustrée en Figure 1.4b, il est clair que la partie processeur/mémoire prend une part extrêmement importante. Cette consommation, provoquée par le CPU d'une ressource de calcul, multipliée par les milliers de machines physiques que regroupe un *data center*, devient alors réellement non négligeable. Le pourcentage de consommation que représente les serveurs au sein d'un *data center* est illustré en Figure 1.4a. Cette problématique de consommation d'énergie représente un réel challenge pour les fournisseurs de services de *Cloud Computing*, car outre le fait de pouvoir montrer que le rendement énergétique de leur *data center* est meilleur que d'autres, cela représente pour eux un coût financier qu'ils ne peuvent ignorer.

La conjonction de ses domaines induit une réflexion poussée sur l'analyse des paramètres de QoS, influençant indéniablement la performance et le rendement des propositions de services, et des conditions d'utilisation de ces services. Du point de vue du fournisseur de services *Cloud* cela provoque donc le choix

22. Étude de Greenpeace publiée en 2012 :
http ://www.greenpeace.org/international/en/publications/Campaign-reports/Climate-Reports/How-Clean-is-Your-Cloud/

de techniques de gestion de leurs infrastructures afin de satisfaire leurs propres intérêts ainsi que ceux de leurs utilisateurs.

Ces domaines de qualité de service et de consommation énergétique constituent le socle des travaux de recherche présentés dans ce manuscrit. La section qui suit présente sous quels angles celles-ci ont été abordées lors des travaux de recherche de ce doctorat.

1.2 Problématiques de recherche

Cette section expose les problématiques de recherche abordées ainsi que les nombreux questionnements que soulève l'étude des deux domaines présentés dans la section précédente. La qualité de service et la réduction de la consommation énergétique peuvent être vues de bien des manières différentes. C'est pourquoi une présentation détaillée des différentes façons dont ces problématiques peuvent être abordées est proposée ci-dessous :

Comment attester de la qualité de service dans les Clouds ?

L'introduction aux problématiques de qualité de service exposée en Section 1.1 exprime l'importance, pour un fournisseur de services, de pouvoir évaluer le niveau de QoS de ces services. Cela, en relation directe avec les définitions des termes contenus dans les SLA, les propriétés des infrastructures *Cloud* et les différents coûts liés au fonctionnement de services *Cloud*. En effet, pouvoir attester du niveau de qualité de service entre deux fournisseurs de services n'est pas chose aisée si les éléments de comparaison entre eux ne représentent pas exactement les mêmes propriétés de leurs infrastructures. Il convient donc d'analyser quels paramètres peuvent être communs entre différentes infrastructures *Cloud*, quels paramètres représentent une propriété physique des infrastructures et quels paramètres dépendent de la manière dont est gérée l'infrastructure. Une harmonisation des paramètres de qualité de service *Cloud*, ayant chacun une signification précise et indépendante des types de services proposés pourrait composer un ensemble de paramètres variés dont l'évaluation conjointe donnerait une estimation normalisée et comparable du niveau de qualité de service des prestations proposées par les fournisseurs de services.

Comment sélectionner les paramètres de QoS à optimiser ?

La qualité de service est un domaine très vaste, pouvant soulever un nombre ingérable de paramètres, à la fois tous liés les uns aux autres mais n'optimisant pas les mêmes propriétés du système et devant être gérés de manières différentes. De plus, la gestion de la consommation énergétique d'un ou plusieurs *data centers* demande également une réelle réflexion car de nombreuses solutions, aidant la diminution des puissances des composants des systèmes, sont envisageables. Cet aspect énergétique est désormais perçu comme un paramètre de qualité de service à part entière par les fournisseurs de services. C'est pourquoi une étude conjointe de ces deux domaines, peut être interprétée comme un problème multi-objectifs, dont la résolution a pour but d'optimiser chacun des paramètres pris en compte. Bien sûr, une telle approche induit le fait que le fournisseur de services accepte de dégrader certains paramètres, non seulement par rapport à leur valeur optimale, mais aussi les uns par rapport aux autres afin de trouver le meilleur équilibre

possible. Une étape importante dans le choix des paramètres de qualité de service est de sélectionner des paramètres ayant un lien direct avec les SLA des fournisseurs. Ainsi, les études utilisant ces paramètres auront un réel intérêt pour le fournisseur et pour ses utilisateurs, mais pourront également servir lors de l'estimation du respect des règles incluses dans les SLA. De plus, il est important que la sélection de ces paramètres regroupe des paramètres étant à la fois antagonistes, variés et pertinents. Cela, afin que la résolution de l'optimisation multi-objectifs puisse fournir des solutions représentant de réels compromis pour le fournisseur de services et n'étant pas déstabilisées par un paramètre qui aurait pour effet de faire tendre toutes les solutions dans une unique direction.

Quelles solutions pour minimiser la consommation énergétique des ressources de calcul ?

Au sein d'un *data center*, les différentes solutions permettant de réduire la consommation d'énergie globale s'appliquent à plusieurs niveaux :

- **Niveau machine physique :** Deux solutions sont à détailler :

  - **Allumage/Extinction (ON/OFF)** : Une manière simpliste de réduire la consommation d'un *data center* est tout simplement de limiter le nombre de machines physiques, et donc d'éteindre celles qui ne sont pas utilisées. Cela dit, quelques conditions sont nécessaires pour pouvoir appliquer cette solution en évitant tout problème. Il est tout d'abord nécessaire de s'assurer que la machine physique en question n'héberge aucun service à cet instant mais aussi que celle-ci n'aura pas besoin d'être utilisée dans les instants à venir. En effet, éteindre et rallumer une machine physique dans un intervalle de temps trop court représente une perte de temps et d'énergie. Une solution intermédiaire est de mettre la machine physique en état d'hibernation (*Suspend-To-RAM* en anglais) [20, 40]. Dans les travaux présentés dans ce manuscrit seuls deux états sont pris en compte : allumé (ON) ou éteint (OFF).
  - **DVFS : Dynamic Voltage & Frequency Scaling** : Le DVFS permet une gestion extrêmement fine de la puissance délivrée par les machines physiques en autorisant l'ajustement des fréquences des processeurs. Ainsi, en fonction de la charge CPU d'une machine physique à un instant t la fréquence CPU peut être ajustée. D'un point de vue énergétique, cette technique se révèle très intéressante en autorisant une diminution de la fréquence CPU (lorsque la charge est faible) et donc des puissances délivrées par les machines physiques concernées.

- **Niveau logiciel :** Le fonctionnement des services de type *Cloud Computing* est basé sur le principe de la virtualisation. Chaque service utilise une ou plusieurs machines virtuelles devant être allouées sur les machines physiques à disposition. La manière d'allouer cette flotte de machines virtuelles influence de nombreux paramètres de qualité et notamment fortement la consommation énergétique de l'infrastructure dans son ensemble. Trois propriétés caractérisant une allocation sont à définir :

  - **"Où"** : Dans un ensemble de machines physiques hétérogènes en termes de puissance délivrée, le choix de la machine physique sur laquelle une machine virtuelle est affectée a son importance.

En effet, une priorité aux allocations faites sur des machines peu consommatrices favorisera la réduction de la consommation d'énergie globale. Un autre effet des allocations peut tendre à surcharger un endroit d'un *cluster* pouvant entraîner une chauffe anormale des machines physiques concernées, et une influence néfaste sur le matériel concerné.

- **"Quand"** : Pouvoir décider de l'instant de départ des exécutions des machines virtuelles peut permettre d'optimiser le temps d'exécution total de celles-ci en trouvant un ordonnancement de départ favorable. En considérant que toutes les machines virtuelles n'ont pas le même temps d'exécution, une allocation intelligente ou une ré-allocation de ces machines peut permettre de réduire le temps d'exécution total et favoriser une réduction de la consommation d'énergie.
- **"Comment"** : La manière dont l'allocation d'une machine virtuelle est effectuée détermine si une reconfiguration de ses capacités de calcul ou de mémoire est autorisée. Réduire la capacité de calcul d'une machine virtuelle revient à diminuer l'espace Mémoire et/ou le pourcentage CPU qui aurait dû lui être fourni. Cette solution permet donc de diminuer les capacités occupées par une machine virtuelle sur une machine physique, ouvrant de nouvelles possibilités d'allocations mais dégradant la qualité d'exécution des machines virtuelles concernées.

Quels seraient les avantages d'une optimisation multi-objectifs QoS Cloud à des fins d'ordonnancement ?

Comme introduit en Section 1.1, la prise en compte de plusieurs paramètres de QoS provoque un conflit d'optimisation entre chacun d'entre eux. De ce conflit inévitable découle un problème de recherche d'optimalité de chaque paramètre. Obtenir une solution permettant d'optimiser de manière parfaite chaque paramètre n'est pas possible [51]. En effet, dès que l'optimisation de deux paramètres, ou plus, tendent à adopter des configurations différentes des ressources physiques, alors le résultat de leur optimisation commune ne peut être optimale. La recherche d'un compromis entre différentes configurations est alors obligatoire. Trouver un compromis entre plusieurs paramètres ne représente pas de difficulté particulière ; trouver un compromis permettant d'obtenir en moyenne (sur l'ensemble des paramètres étudiés) un meilleur résultat que toutes les autres solutions possibles est par contre un réel enjeu. Afin d'être en mesure de pouvoir estimer la qualité d'un compromis par rapport à un autre, une solution serait de regarder les variations de chaque paramètre de QoS provoquées par différentes méthodes d'optimisation. Une autre manière de procéder peut être de trouver l'optimal de chacun des paramètres pris en compte, puis de regarder le pourcentage de dégradation que provoque une optimisation multi-objectifs. S'il est impossible de connaître la valeur optimale d'un paramètre, par exemple parce que celle-ci n'existe pas (pour le cryptage de données par exemple), une autre façon d'estimer la qualité d'un compromis est de comparer les valeurs des paramètres données par celui-ci par rapport aux valeurs des paramètres données par d'autres solutions (non-optimales) résultant d'une optimisation unique de chaque paramètre. Une telle comparaison peut permettre d'estimer la dégradation des paramètres entre les différents compromis obtenus, et ainsi d'estimer la qualité d'une optimisation multi-objectifs.

Comment analyser l'impact d'une optimisation multi-objectifs sur l'ordonnancement ?

L'analyse des résultats d'une optimisation multi-critères de QoS *Cloud* nécessite de pouvoir procéder à un processus cyclique comprenant :

- Des mesures de métriques
- Le stockage et l'analyse de leur valeur
- Une prise de décision de ré-allocation en fonction des résultats
- L'exécution d'un ordonnanceur associé à un algorithme
- La mise en place du nouveau placement

Toutes ces étapes, bien que relativement simples séparément les unes des autres, ne sont pas facilement réalisables sur de réelles plates-formes. En effet, les expérimentations réelles sont souvent limitées à cause des équipements physiques de mesures inclus au sein des infrastructures. Cela est d'autant plus restreint lorsque les études intègrent des mesures de paramètres que les équipements ne permettent pas. Une solution alternative intéressante est d'effectuer ces phases d'évaluation par simulation. En effet, plusieurs simulateurs de *Cloud* sont actuellement développés et utilisés. Il est évident que chaque simulateur n'intègre pas les mêmes caractéristiques, tant en termes de fonctionnalités qu'en perspectives d'évolutions. Le choix de celui-ci est donc crucial afin de pouvoir concevoir de nouveaux outils (gestion des machines virtuelles, politiques d'ordonnancement, modèles énergétiques et gestion des fréquences CPU par exemple) répondant aux critères des travaux de recherche. Enfin, l'impact d'une optimisation multi-objectifs doit pouvoir être analysé au cours du temps. Soit pour être en mesure de prendre des décisions au cours du temps, soit simplement pour avoir une représentation temporelle des évolutions des différentes métriques de qualité de service. En effet, les comparaisons des résultats d'une optimisation multi-objectifs par rapport à d'autres approches doivent pouvoir être effectuées non seulement dans différentes configurations d'optimisation, mais également dans différentes conditions d'utilisation des ressources physiques (variations de charge du système). Une solution est d'opter pour une analyse par simulation, permettant d'avoir un aperçu temporel de l'évolution des différents paramètres analysés. L'utilisation d'un simulateur engendre des problématiques de validation des différentes fonctionnalités de celui-ci par rapport au comportement d'une réelle plate-forme, cela pour garantir la validité des résultats obtenus. Malgré la difficulté à effectuer des expérimentations sur de réelles plates-formes, il est primordial de favoriser celles-ci lorsque la situation le permet. De plus, outre le fait que les résultats d'expérimentations ont une légitimité théoriquement sûre, ceux-ci peuvent également être utilisés afin de calibrer et de valider le bon comportement d'un simulateur. Cela, notamment en termes de modèle énergétique et de gestion dynamique de fréquence CPU.

1.3 Contributions scientifiques

Cette section expose les trois contributions découlant de l'ensemble des travaux de ce doctorat.

Modélisation de QoS dédiée au Cloud Computing

Après analyse de l'état de l'art concernant les études de qualité de service pour le *Cloud Computing*, il apparaît que la qualité de service est fréquemment modélisée par des métriques basiques, néanmoins intéressantes, tel que le temps de réponse, la consommation énergétique ou le coût financier à la charge des fournisseurs de services. En analysant les clauses des contrats SLA des fournisseurs de services actuels il ressort que bien des paramètres ne sont pas pris en considération dans les études actuelles de QoS en environnement *Cloud* dédiées à améliorer l'ordonnancement de machines virtuelles. La première contribution de cette thèse repose sur une analyse large de multiples paramètres de qualité de service. C'est ainsi que l'ensemble des paramètres de QoS proposés sont répertoriés en quatre catégories distinctes, incluant pour chacun d'entre eux une définition de leur signification et de leur intérêt, associées à une métrique mesurable et ré-utilisable lorsque ceux-ci le permettent.

Enrichissements des approches multi-objectifs de QoS Cloud

L'étude de la modélisation de la qualité de service dédiée aux services de *Cloud Computing* a permis de définir des métriques, mesurables et réutilisables, pouvant ainsi être intégrées dans n'importe quelles heuristiques ou algorithmes d'allocation de machines virtuelles. Les algorithmes de placement présentés dans les études actuelles se limitent généralement à l'étude de deux ou trois métriques qui sont bien souvent les mêmes. Ce listing de paramètres et de métriques de qualité de service permet d'élargir l'espace des métriques évaluées lors des processus de décision de ces algorithmes. Aussi, celles-ci permettent de se rapprocher des paramètres définis dans les SLA des fournisseurs de services *Cloud* actuels. Une sélection de paramètres de qualité de service dans les *Clouds* a poussé à implémenter un algorithme génétique (GA) spécifique au problème d'ordonnancement de machines virtuelles de services *Cloud*. Cet algorithme génétique, détaillé en Section 4.4, intègre deux métriques habituellement ignorées : la robustesse et le dynamisme, ainsi que deux métriques plus communément utilisées : la consommation énergétique et le temps de réponse. Une analyse du comportement de l'algorithme génétique face à ce problème d'optimisation multi-critères a pu être établie en comparant les valeurs de ces métriques antagonistes obtenues dans différentes configurations d'optimisation. En effet, différentes versions mono et multi-objectifs ont été évaluées afin d'analyser l'influence de chacune de ces métriques et démontrer l'intérêt de l'utilisation d'une optimisation multi-objectifs. Un autre aspect de cette contribution est de montrer la pertinence de certains algorithmes, aussi basiques soient-ils, sur des métriques non négligeables aux yeux d'un fournisseur de services.

Simulations d'ordonnancement sous contraintes de QoS Cloud

L'évaluation de l'impact d'une optimisation multi-objectifs de paramètres de qualité de service de *Cloud* a été réalisée par simulation. Une étape importante de thèse a été d'étendre les fonctionnalités d'un simulateur de *Cloud Computing*, CloudSim. Tout d'abord, une amélioration de ce simulateur a consisté à

intégrer l'outil de gestion dynamique de fréquences CPU, le DVFS, pour pouvoir simuler le comportement réel de cet outil inclus dans le noyau Linux depuis plus de 10 ans. L'intégration du DVFS représente un réel atout en termes de simulation énergétique, celui-ci permettant un ajustement des fréquences CPU et donc de la consommation énergétique des machines physiques très précis. Aussi, la validation du bon comportement de cette nouvelle fonctionnalité d'une granularité très fine a nécessité la mise en place de plusieurs campagnes d'expérimentations sur de réelles plates-formes. Ces expérimentation servant à la fois à comparer les résultats des simulations, mais aussi à calibrer le simulateur notamment en termes de modèles énergétiques. Les plates-formes utilisées sont les suivantes :

- Grid'5000 [23] : Grille de calcul française dédiée aux travaux de recherche, répartie sur 10 sites en France.
- CalMip [24] : Supercalculateur Hyperion, situé au CICT (Centre Inter-universitaire de Calcul de Toulouse).
- RECS [25] : Plate-forme dédiée aux études thermiques des processeurs et aux consommations d'énergie, située à l'IRIT (Institut de Recherche en Informatique de Toulouse).

De plus, une évaluation des paramètres de QoS a été intégrée au simulateur, afin que celui-ci puisse fournir des mesures précises de l'ensemble des paramètres de QoS à chaque pas de simulation.

La combinaison de cet ensemble de fonctionnalités mis en jeu lors des simulations effectuées a permis d'analyser l'évolution de plusieurs paramètres de qualité de service au cours du temps, cela en utilisant différents algorithmes d'ordonnancement.

Les différentes évolutions apportées au simulateur CloudSim ont conduit à une riche collaboration avec le laboratoire au sein duquel ce simulateur est développé : le *Cloud Computing and Distributed Systems (CLOUDS) Laborartory* de Melbourne, dirigé par Prof. BUYYA Rajkumar. La version du simulateur présentée dans [61], intégrant le DVFS, est téléchargeable sur la page d'accueil [26] du site internet de ce laboratoire, ou directement à cette adresse [27].

1.4 Plan du manuscrit

Suite à cette introduction, ce manuscrit s'articule en 6 autres chapitres :

- **Chapitre 2 : État de l'art**
  Ce chapitre présente de manière assez large, illustrés par des exemples ou des figures, les différents domaines abordés de plus ou moins loin dans cette thèse. Tout d'abord, le principe de virtualisation étant intrinsèquement lié au *Cloud Computing*, une présentation des différentes techniques de virtualisation est proposée. Suit une présentation des concepts de complexité et des différents types d'algorithmes et méta-heuristiques pouvant être utilisés à des fins d'ordonnancement. Pour coller à l'un des principaux domaines de cette thèse, une synthèse des outils et des travaux en relation avec

23. http ://www.grid5000.fr
24. http ://www.calmip.cict.fr
25. http ://recs.irit.fr
26. http ://www.cloudbus.org/cloudsim
27. http ://www.cloudbus.org/cloudsim/CloudSim_DVFS.rar

la gestion des consommations d'énergie dans les *Clouds* fait l'objet d'une troisième section. Suit une analyse des propositions de SLA des fournisseurs de services *Cloud* actuels afin de mettre en avant les réels paramètres de QoS utilisés dans les SLA. Enfin, un recensement des simulateurs de *Cloud Computing* détaillant les caractéristiques et les propriétés de chacun termine ce chapitre.

- **Chapitre 3 : Modélisation : de l'Architecture à la Qualité de Service**
  Ce chapitre est dédié à la présentation de l'ensemble des modèles, d'architecture et de QoS, définis et utilisés dans l'ensemble des travaux. Tout d'abord, une explication des notations utilisées dans l'ensemble des modèles est proposée afin d'aider à la lecture de ceux-ci. Une seconde section contient la modélisation de l'architecture matérielle et logicielle de l'environnement *Cloud* considéré. Enfin, une modélisation de paramètres de qualité de service dédiée au *Cloud Computing* est exposée, intégrant explications, définitions et propositions de métriques.

- **Chapitre 4 : Optimisation multi-objectifs de qualité de service Cloud**
  Ce chapitre expose pourquoi et comment la modélisation de paramètres de QoS *Cloud* a pu être utilisée afin de mettre en avant l'intérêt d'une approche d'ordonnancement multi-objectifs en environnement *Cloud*. Les enrichissements que représente une telle approche sont tout d'abord exposés. Puis, les détails des métriques de QoS sélectionnées, des différents algorithmes et de la méta-heuristique (algorithme génétique) mis en jeu sont décrits. Enfin, une analyse des résultats de différentes configurations d'optimisation de l'algorithme génétique est proposée.

- **Chapitre 5 : CloudSim : simulations énergétiques avec DVFS**
  Ce chapitre présente le simulateur CloudSim utilisé tout au long de cette thèse, ainsi que les améliorations qui lui ont été apportées et la validation de leur bon fonctionnement. Ainsi l'implémentation du DVFS et sa validation par différents cas d'utilisation sont présentés, comparant les résultats d'expérimentations menées sur de réelles plates-formes, à celles du simulateur.

- **Chapitre 6 : Simulations d'ordonnancement *Cloud* sous contraintes de QoS**
  Ce chapitre expose les différents résultats obtenus lors des phases d'évaluation avec le simulateur CloudSim, intégrant les métriques de QoS et les algorithmes d'allocation présentés au Chapitre 4. L'aspect temporel apporté par les simulations permet ainsi d'analyser l'évolution des différentes métriques de QoS suivant les algorithmes d'allocation utilisés.

- **Chapitre 7 : Conclusions et Perspectives**
  Ce dernier chapitre apporte une vision globale de l'ensemble des travaux réalisés lors de cette thèse, ainsi que les perspectives d'évolution et d'amélioration des résultats de recherche pouvant être envisagées lors de futurs doctorats.

2

État de l'Art

Sommaire

Ce chapitre d'État de l'Art propose un aperçu de cinq domaines liés au *Cloud Computing*, tous abordés au cours de cette thèse. Ces domaines sont à la fois distincts les uns des autres, ayant chacun leur propre utilité, mais se doivent d'être utilisés conjointement dans les recherches actuelles afin de proposer des études complètes.

Les sections de ce chapitre proposent :

- Une présentation du concept de virtualisation, incluant ses différentes évolutions et configurations de fonctionnement.
- Une analyse de multiples algorithmes et méta-heuristiques pouvant être utilisés pour l'allocation et l'ordonnancement de machines virtuelles.
- Le détails des outils de gestion de consommation de ces centres de calcul, et une analyses des études dédiées à la qualité de service (problèmes énergétiques inclus) dans le cadre du *Cloud Computing*.
- Une présentation des différentes propositions de contrat SLA de fournisseurs de services *Cloud* actuels tel que Microsoft Azure, Amazon et Oracle.
- Des résumés détaillés des différents simulateurs de *Cloud Computing* utilisés de nos jours, expliquant leurs fonctionnalités, leurs avantages et inconvénients en termes d'architecture interne, d'outils de

gestion de consommation énergétique, mais aussi de possibilités d'évolutions.

Voici comment s'articulent entre eux ces différentes domaines composant les différentes sections de chapitre au sein des travaux présentés dans ce manuscrit :

Le *Cloud Computing* repose directement sur le concept de virtualisation. La virtualisation permet d'apporter une couche d'abstraction entre le matériel physique utilisé et les différents systèmes d'exploitation et applications que l'on souhaite utiliser. En effet, l'utilisation de machines virtuelles permet de faire fonctionner de faire fonctionner en parallèle plusieurs systèmes sur une même machine physique, principe indispensable pour mettre en place le fonctionnement des services au sein d'un *Cloud*. Du point de vue du fournisseur de service, toutes ces machines virtuelles sont à répartir sur l'ensemble des machines physiques dont il a à sa disposition. Afin de répartir celles-ci de manière intelligente et efficace, l'utilisation d'algorithmes de placement est indispensable. Ces algorithmes permettent d'optimiser, par exemple pour des raisons de performance ou de réduction des coûts énergétiques, l'allocation de ces machines virtuelles. Ainsi, la deuxième section de ce chapitre propose un aperçu de divers algorithmes ou méta-heuristiques détaillant leurs comportements ainsi que leurs avantages et inconvénients. Les serveurs de *Cloud Computing* pouvant héberger des milliers de machines virtuelles simultanément, les phases d'ordonnancements de celles-ci sur l'ensemble des *data centers* et machines physiques disponibles ont une grande importance. Cela, afin d'assurer et/ou améliorer la qualité de service de l'ensemble de la plate-forme. De nos jours, tout cela constitue un réel domaine de recherche alliant l'utilisation d'outils visant à améliorer le rendement global des systèmes. Ces techniques concernent à la fois la gestion des machines virtuelles mais aussi l'utilisation d'outils ayant une influence directe sur la consommation énergétique des machines physiques. Ces différentes méthodes d'ordonnancement et de gestion énergétique jouent un rôle prépondérant sur le niveau de qualité de service proposée aux utilisateurs des services. Ainsi, les fournisseurs de services peuvent tirer profit de ces différentes approches afin de gérer la qualité de service de l'ensemble de leur infrastructure, mais également afin de tenir les engagements pris dans les contrats de SLA. C'est pourquoi, une présentation des termes de ces contrats SLA, provenant de réels fournisseurs de services *Cloud* actuels est proposée afin de comprendre et d'analyser quels paramètres rentrent en jeu, et comment ceux-ci peuvent être inclus dans les travaux de recherche de façon plus précise qu'actuellement. Enfin, la dernière section de chapitre concerne le simulateurs de *Cloud* actuellement disponibles. En effet, afin de conduire des campagnes d'études et/ou d'évaluations de performance de nouvelles modélisation de qualité de service dédiées au *Cloud Computing* il est malheureusement très difficile de mener celles-ci sur de réelles plates-formes. L'utilisation de simulateurs est très intéressant, dans la mesure où ceux-ci intègrent les outils nécessaires aux études de nouvelles approches. Cette section de comparaison de simulation permet donc d'évaluer les différents outils qu'ils proposent (virtualisation, gestion des machines physiques, modèles énergétique et de QoS, etc...) ainsi que leur niveau de flexibilité reflétant la difficulté avec laquelle l'ensemble des fonctionnalités nécessaires pour les travaux présentés dans ce manuscrit peuvent y être intégrés.

La première section de ce chapitre présente le concept de virtualisation.

2.1 La virtualisation

Cette section introduit le principe de virtualisation qui est la base du fonctionnement des services de type *Cloud Computing*. Celui-ci permet le partage des ressources physiques disponibles dans les *data centers*, le démarrage, l'arrêt, la gestion des servies mis à disposition des utilisateurs ainsi que plusieurs outils : gestion énergétique ou isolation des données utilisateurs par exemple.

Son apparition, ses utilisations, son application sur les différents composant d'une machine physique ainsi que les différentes architectures de virtualisation existantes composent les différentes parties de cette section.

Il faut remonter au cours des années 1960 pour voir apparaître les prémices des premières techniques de virtualisation [117]. Les ordinateurs centraux de IBM étaient de grands systèmes multi-processeurs. Ils étaient conçus pour permettre de haut débit d'entrée/sortie (E/S), une sécurité accrue et une fiabilité à toutes épreuves. Ils étaient généralement installés dans les grandes entreprises ou des organisations gouvernementales. Pendant cette période IBM a été confronté aux premiers besoins de partitionnement de leurs ordinateurs en plusieurs systèmes d'exploitation afin de répondre aux besoins de utilisateurs qui utilisaient des applications développées pour différents systèmes d'exploitations [9].

2.1.1 Techniques et Architectures de virtualisation

Notion de Virtual Machine Monitor

La première solution proposée par IBM fut un couche applicative, dénommée *Virtual Machine Monitor* (VMM) ou Hyperviseur, située entre la couche physique et les systèmes d'exploitation. Ainsi, chaque système d'exploitation s'exécute au-dessus de l'hyperviseur, ce dernier ayant pour rôle de contrôler l'accès aux ressources physiques de chaque système d'exploitation (Figure 2.1).

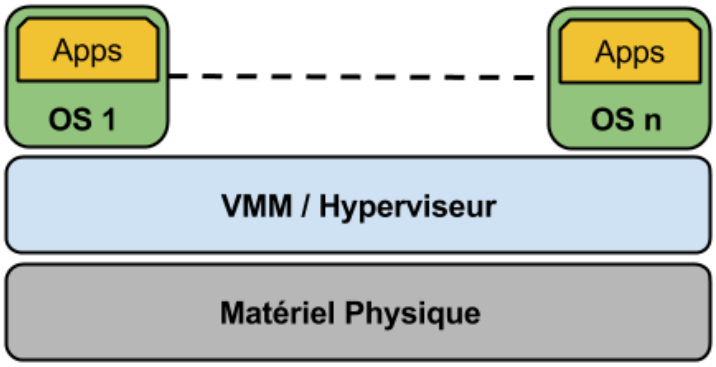


Figure 2.1 – Architecture de virtualisation VMM.

La virtualisation selon VMWare

Comme présentées en introduction à cette thèse, la révolution *Internet* et l'apparition d'utilisation de serveurs et d'ordinateurs personnels, de plus en plus puissants par les entreprises au cours des années 1990

ont engendré l'utilisation de milliers de machines physiques, gourmandes en énergie et n'étant pas toujours utilisées à 100% de leur capacité. C'est alors que de nouvelles méthodes de virtualisation des ressources vont voir le jour, afin d'optimiser la manière de les utiliser, de profiter au mieux des capacités de ces serveurs et donc d'améliorer leur rendement. C'est alors au tour de VMWare [135] de s'attaquer au concept de virtualisation des serveurs x86 (dénomination de l'architecture des processeurs 32 bits utilisés en cette période), permettant d'exécuter plusieurs machines virtuelles sur une seule machine : principe désormais connu sous le nom de "consolidation" de machines virtuelles. Chaque machine virtuelle peut être démarrée, stoppée ou suspendue et est définie par ses besoins en ressources CPU/Mémoire et utilise une image du disque (noyau, applications et librairie) de la machine physique. Ce type de virtualisation reprend la base de l'architecture proposée par IBM en utilisant un Hyperviseur entre la couche physique et la/les couches virtuelles, mais ce dernier permet désormais de donner l'impression à chaque machine virtuelle qu'elle est exécutée et qu'elle peut utiliser directement la couche physique. Ce fonctionnement, autorisé par un multiplexage de la couche physique demande également une forte isolation entre chaque machine virtuelle ainsi que des mécanismes de sécurité au sein de l'hyperviseur, afin d'éviter tout conflit d'accès à la couche physique (accès mémoire notamment). Cette architecture de virtualisation est représentée en Figure 2.2.

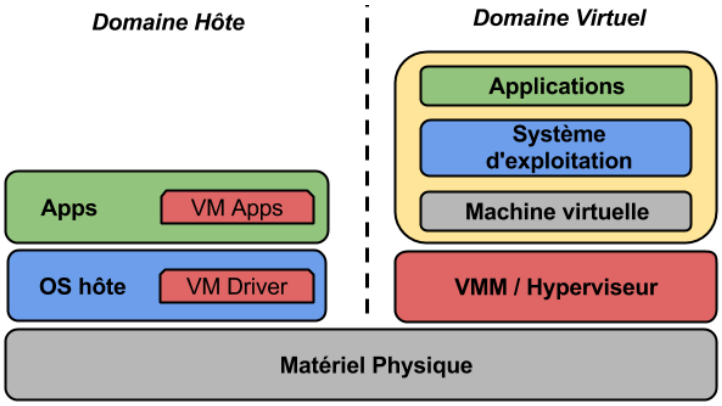


Figure 2.2 – Architecture de virtualisation par VMWare.

Virtualisation vs Émulation

La virtualisation ne doit pas être confondue avec l'émulation. Contrairement à la virtualisation qui implique le multiplexage de la couche physique entre plusieurs systèmes d'exploitation "invités" créant alors l'illusion pour ces derniers de s'exécuter au-dessus de la couche physique réelle, l'émulation fournit aux systèmes d'exploitation invités un environnement physique (CPU, mémoire, E/S) entièrement émulé par une couche logicielle. S'il on prend un exemple concret, dans un environnement virtualisé le système d'exploitation des machines virtuelles doit être en mesure de fonctionner en utilisant l'architecture de processeur (par exemple x86) de la machine physique sur laquelle elles sont hébergées car leurs instructions CPU sont réellement exécutées par le processeur de la machine physique hôte. En revanche, l'émulation permet d'exécuter n'importe quel système d'exploitation, développé ou non pour l'architecture du processeur de la machine physique hôte. Ainsi, des systèmes d'exploitation spécifiques à des architectures processeurs peuvent être utilisés au-dessus d'un système d'exploitation dédié à une architecture x86 par

exemple. L'émulation ne donne pas un accès direct à la couche physique aux systèmes d'exploitation "invités", mais l'accès au CPU ou la Mémoire est géré par l'émulateur. Ainsi, le démarrage, l'arrêt, le clonage ou la migration de systèmes d'exploitations émulés est géré par le système d'exploitation de la machine physique hôte. La représentation de l'utilisation de systèmes d'exploitation est représentée en Figure 2.3.

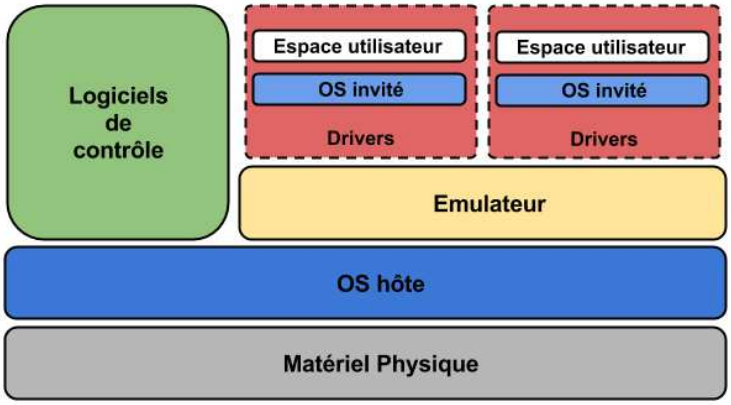


Figure 2.3 – Architecture d'une utilisation de systèmes d'exploitations spécifiques par émulation.

La para-virtualisation

La para-virtualisation (ou virtualisation de type 1), fonctionne avec un hyperviseur (ou VMM) juste au-dessus de la couche matérielle. Cet hyperviseur nécessite l'installation d'un noyau spécifique, le plus connu étant "Xen Paravirtualization" [15]. Ce type d'architecture de virtualisation nécessite d'introduire une priorité dans les instructions, représenté par la figure 2.4.

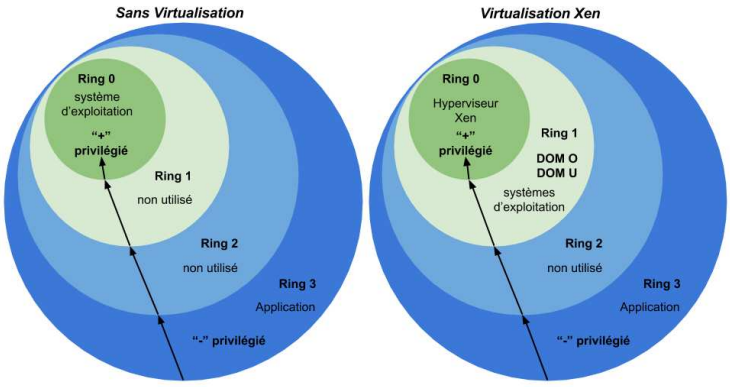


Figure 2.4 – Représentation des niveaux de priorité d'accès (des instructions) aux ressources physiques

Les niveaux de privilèges sont un concept fondamental intégré dans les processeurs modernes pour assurer une certaine sécurité dans les systèmes d'exploitations. En particulier, les niveaux de privilèges permettent au système d'exploitation d'empêcher les utilisateurs et/ou les processus, avec des niveaux de privilèges différents, d'accéder sans contrôle aux ressources physiques partagées que le processeur, la mémoire et l'accès au disque (E/S). La gestion du contrôle d'accès aux ressources permet donc d'éviter

tout problème de conflit mais régule aussi l'utilisation des ressources partagées en respectant le niveau de privilège de chaque processus, évitant ainsi tout risque d'accès non sécurisé aux ressources physique. En effet, ce travail est donné au système d'exploitation, qui a donc la responsabilité d'accorder ou refuser l'accès au matériel physique et d'exécuter les requêtes des processus utilisateur (par exemple, envoi de données via le réseau), en inter-action avec le matériel physique.

Comme on peut le constater sur la Figure 2.4, il existe quatre niveaux de privilèges : le niveau 0 bénéficie du privilège le plus haut, inversement, le niveau 3 bénéficie du privilège le plus faible, pour accéder au matériel physique. De nos jours, les noyaux de la plupart des systèmes d'exploitation modernes tel que Linux ou Windows fonctionnent dans le niveau de privilège le plus élevé, alors que les processus utilisateur sont généralement exécutés dans le niveau de privilège le plus bas. Les niveaux de privilèges intermédiaires restent inutilisés. En d'autres termes, si un processus utilisateur s'exécute sur le CPU, le CPU est en niveau de privilège le plus bas. De ce fait, lorsqu'un processus utilisateur charge le système d'exploitation (par interruption logicielle) d'exécuter des instructions privilégiées (par exemple des accès E/S), alors le processeur prend les droits du niveau de privilège le plus élevé afin de pouvoir exécuter ces instructions.

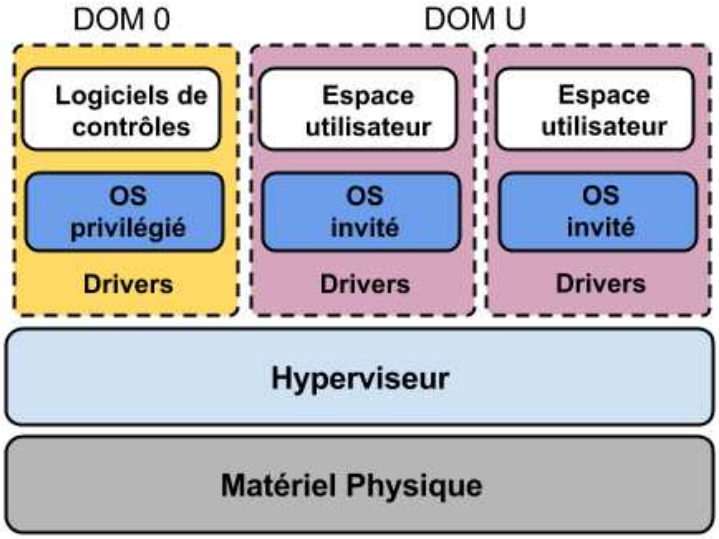


Figure 2.5 – Architecture d'une para-virtualisation.

Dans la para-virtualisation de type Xen, l'hyperviseur s'exécute au niveau de privilège le plus élevé. L'hyperviseur Xen est un noyau léger et ne met pas la mise en œuvre d'une gestion complexe de décisions (contrôle d'admission, ordonnancement des machines virtuelles par le CPU). C'est pourquoi, une interface de commande de bas niveau, capable de prendre de telles décisions, est utilisée. La para-virtualisaton Xen intègre donc deux domaines différents : "DOM 0" et "DOM U". Le "DOM 0" est le domaine qui est lancé au démarrage de la machine physique, dans lequel le noyau Xen s'exécute. Ce domaine reçoit donc le niveau d'accès aux ressources physiques le plus privilégié ce qui permet au noyau Xen de communiquer directement avec le matériel physique via les pilotes du noyau. C'est aussi dans ce domaine que s'exécute le système d'exploitation dit "privilégié", contenant les fichiers de configurations des systèmes d'exploitation

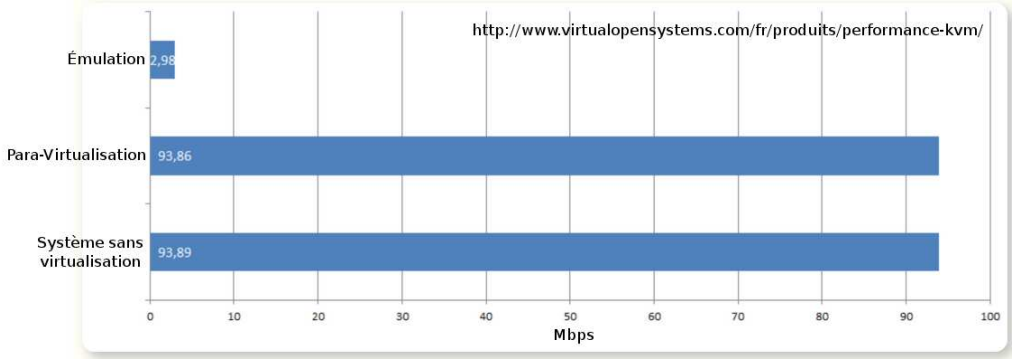


Figure 2.6 – Comparaison de performance réseau entre l'émulation et la para-virtualisation.

appartenant au "DOM U ", et qui a aussi pour rôle de contrôler le cycle de vie des systèmes d'exploitations "invités" s'exécutant dans le "DOM U".

Comme l'hyperviseur Xen occupe le niveau de privilège le plus élevé (niveau 0), les systèmes d'exploitation "invités" de sont pas en mesure de fonctionner (accès aux ressources) de façon indépendante. Alors, les systèmes d'exploitation "invités" ("DOM U") s'exécutent au niveau de privilège 1, tout comme le "DOM 0". Ainsi, toutes instructions nécessitant un niveau de privilège élevé provenant des systèmes d'exploitation "invités" sont délégués à l'hyperviseur Xen, par l'intermédiaire d'*hypercall* (Figure 2.8). Ces *hypercall* sont capturés par l'hyperviseur qui peut ainsi exécuter les instructions provenant du niveau de privilège 1.

De plus, la para-virtualisation peut donner de très bons résultats en termes de performance, notamment en comparaison avec l'émulation comme illustré en Figure 2.6. Des bibliothèques tel que Virtio [118] peuvent aussi être utilisée afin d'optimiser les performances des entrées/sorties.

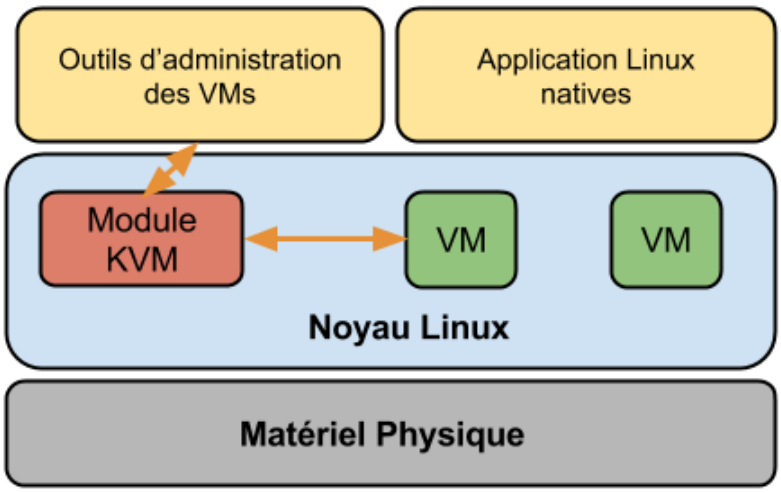


Figure 2.7 – Architecture d'une virtualisation totale tel que KVM dans le noyau Linux

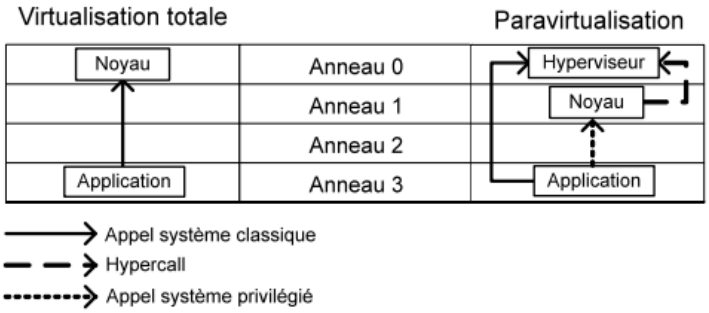


Figure 2.8 – Comparaison du fonctionnement des appels systèmes entre un environnement de virtualisation totale et une para-virtualisation.

KVM : La virtualisation totale

Enfin, la dernière technique de virtualisation présentée dans cette section concerne la virtualisation totale (*Full Virtualization*), représentée en Figure 2.7 par l'utilisation de KVM [75] (*Kernel-based Virtual Machine*) [1]. Cette solution de virtualisation totale fonctionne dans le noyau Linux et est dédiée aux processeurs x86 intégrant les extensions "Intel VT" ou "AMD-V" [151]. Ces technologies d'accélérations matérielles sont mises à profit pour permettre la virtualisation de matériels physiques. L'activation de ces technologies intégrées aux processeurs x86 d'aujourd'hui se fait par le chargement d'un module KVM de base "kvm.ko" et d'un module spécifique au type de processeur utilisé, Intel ou AMD, nécessitant respectivement le chargement des modules "kvm-intel.ko" ou "kvm-amd.ko". Alors, toute machine virtuelle démarrée est vue par le système comme un processus quelconque, peut accueillir n'importe quel type d'image de système d'exploitation non modifiée et bénéficie d'un environnement physique virtualisé privé. Ce composant KVM est inclus dans le noyau Linux depuis la version 2.6.20.

Une comparaison du fonctionnement des appels système entre une virtualisation totale de type KVM et une para-virtualisation est montrée en Figure 2.8.

2.2 Complexité, Algorithmes Gloutons, Méta-Heuristiques, Programmation linéaire

Cette section présente de manière résumée, le principe de complexité des algorithmes d'ordonnancement, puis détaille (avec des exemples d'algorithmes) les trois grandes catégories d'approches existantes : les algorithmes gloutons, les méta-heuristiques et l'approche par programmation linéaire.

De multiples informations supplémentaires et complémentaires à celles présentées dans cette section sont expliquées de façon très poussée dans la thèse de Hélène TOUSSAINT [137].

1. http://www.linux-kvm.org/page/Main_Page

2.2.1 Problème et Complexité

La théorie de la complexité [28] fournit des outils d'évaluation, à priori et à posteriori des performances des algorithmes et de la difficulté des problèmes. Elle permet d'estimer, le temps et les besoins (par exemple de mémoire) nécessaires à la résolution d'un problème. La théorie de la complexité permet donc de classer (en un certain nombre de classes) les problèmes et d'établir des relations d'inclusion parmi ces classes, appelées "classes de complexité". Étant donné qu'un problème peut être résolu par l'utilisation de différents algorithmes, ayant donc des complexités différentes, il convient de prendre en compte l'algorithme ayant la complexité la plus faible et capable de résoudre le problème pour ainsi définir la complexité du problème considéré.

Afin d'introduire les différentes classes de complexité, la définition d'un problème *décidable* se doit d'être introduite :

- Un problème P est dit *décidable*, s'il existe un algorithme qui résout P en un temps fini. Si aucun algorithme ne peut résoudre P en un temps fini, alors P est dit *indécidable*. Parmi les problèmes décidables, trois classes de complexité sont définies :

Définition 2.1 : Classe P

Un problème P est dit polynomial (ou polynomial-temps), s'il existe un algorithme de complexité polynomiale capable de résoudre P. Ainsi, l'ensemble des problèmes polynomiaux forme la classe P.

Définition 2.2 : Classe NP

Un problème P est dans NP si pour toute instance de ce problème, il est vérifiable en temps polynomial que celle-ci est solution du problème. Cette classe de complexité contient la plupart des problèmes d'optimisation combinatoire, et il est toujours possible de vérifier en temps polynomial, pour n'importe quelle instance d'un problème polynomial, si celle-ci est solution de ce problème.

Ces classes de complexité sont résumées en Figure 2.9, montrant les relations d'inclusions entre celles-ci.

Définition 2.3 : Problème NP-Complet

Une notion essentielle pour définir ce qu'est un problème NP-complet est celui de la dominance : Un problème P domine un problème P' (ou encore P' est réductible polynomialement en P) si :

- On peut transformer toute instance I de P en une instance I' de P' grâce à un algorithme polynomial.
- I est solution de P si et seulement si I' est solution de P'.

Il découle alors de cette définition qu'un problème NP-complet est un problème de la classe NP qui domine tous les autres. La notion de NP-complétude est donc associée à la notion de dominance exprimée ci-dessus. Encore aujourd'hui, aucun algorithme n'est capable de résoudre un tel problème en temps po-

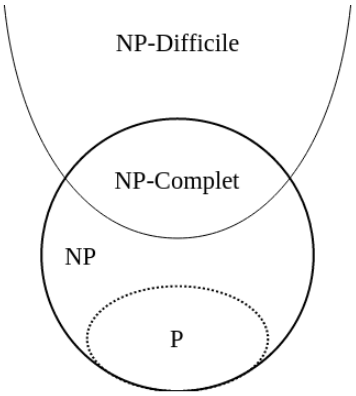


Figure 2.9 – Illustration des inclusions des classes de complexité.

lynomial. Ainsi, différents types d'algorithmes sont employés afin de résoudre de tels problèmes :

- les algorithmes approchés (appelés heuristiques), qui permettent de trouver des solutions approchées dans des temps de calcul raisonnables, mais sans avoir d'indication sur la qualité de la solution trouvée.
- les algorithmes d'approximation avec garantie, qui permettent de trouver des solutions approchées et garantissent une qualité de la solution fournie (sous forme d'écart maximal à la solution optimale) pour toutes les instances du problème.
- les algorithmes exacts, comme l'exploration arborescente ou la programmation linéaire qui, couplées avec des mécanismes de filtrage, peuvent s'avérer très performantes. Ces méthodes fournissent des solutions exactes mais le temps de calcul n'est pas borné polynomialement. C'est pourquoi, elles sont souvent réservées à des instances de taille modérée.

2.2.2 Algorithmes Gloutons

Les algorithmes gloutons (*greedy algorithms* en anglais) sont parmi les schémas heuristiques les plus simples et les plus rapides. Ils construisent une solution de manière itérative sans jamais remettre en cause les décisions prises à l'itération antérieure. Ces algorithmes construisent une solution élément par élément. A une itération donnée, on détermine l'élément à inclure dans la solution partielle en évaluant le coût de la nouvelle solution partielle qui inclut cet élément. L'élément engendrant la plus petite augmentation du coût est choisi. Cependant, dans le cas général, ces algorithmes sont approchés et les insertions qui semblent les meilleures à une itération donnée peuvent s'avérer, en fait, inappropriées pour la suite du processus. Malheureusement, il n'est pas possible de connaître, a priori, l'impact des décisions prises à une itération donnée sur le long terme. En outre, il est possible qu'à une itération donnée aucun élément ne soit insérable (par exemple à cause de contraintes qui se retrouvent violées), dans ce cas, l'algorithme échoue.

2.2.3 Méta-Heuristiques

Une méta-heuristique, contrairement à une heuristique simple qui elle désigne un algorithme qui résout un problème d'optimisation donné sans garantie d'optimalité mais dans des temps de calcul raisonnables, désigne un schéma algorithmique général qui peut s'appliquer à différents problèmes d'optimisation combinatoire. Plus précisément, elle utilise des stratégies guidant la recherche dans l'espace des solutions. Ces stratégies sont donc indépendantes du problème auquel on les applique. Le but est d'explorer le plus efficacement possible l'espace des solutions afin de ne pas rester bloqué dans les minima locaux et de se diriger rapidement vers les régions les plus prometteuses. Il existe un grand nombre de méta-heuristiques allant de schémas très simples (qui mettent en œuvre des processus de recherche basiques, comme la descente, à des schémas beaucoup plus complexes (avec des processus de recherche élaborés comme les colonies de fourmis).

Problèmes intrinsèques aux Méta-Heuristiques :

- **Minima locaux** : Afin de ne pas stagner dans un minimum local, une méta-heuristique doit accepter une dégradation temporaire de la solution en utilisant des opérations qui augmentent le coût de la solution courante. Pour éviter une trop grande divergence de ce procédé, il convient de mettre en œuvre un mécanisme de contrôle. Cependant, il est difficile de prévoir à quel point la solution doit être dégradée pour sortir du minimum local, ce qui rend le réglage de ce mécanisme très délicat
- **Opérateurs de transformations locales** : Des transformations locales différentes peuvent changer complètement le voisinage d'une solution et sont donc des outils d'une grande importance dans ce type d'approche. Cependant, leur application n'est pas déterministe. En effet, ces opérations ne garantissent pas que les nouvelles solutions obtenues soient toujours meilleures que la solution dont elles sont issues. Elles peuvent donc favoriser ou non la convergence de la méthode vers de meilleures régions de l'espace des solutions.
- **Paramétrages** : Les méta-heuristiques sont paramétrées (toutes ont au moins un critère de fin qu'il faut déterminer), le réglage se fait généralement de manière expérimentale même s'il existe des résultats théoriques pour certaines méta-heuristiques [47].
- **Génération des solutions de départ** : Cette étape est indispensable à toute méta-heuristique. Bien sûr, plus les solutions initiales au problème étudié seront bonnes, plus la méta-heuristique aura de chance de visiter des régions de l'espace des solutions intéressantes et donc de converger plus rapidement vers une solution acceptable.

Recuit simulé

Le recuit simulé est une des méta-heuristiques les plus connues (développée simultanément par Kirkpatrick et al. [74] et Cerny [140]) incluant l'acceptation de transformations dégradant le coût de la solution courante. Elle tire son nom du domaine de la métallurgie [139]. Cette méta-heuristique est caractérisée par la présence d'une variable de contrôle appelée température (par analogie aux processus thermodynamiques dont elle s'inspire) qui fixe les conditions dans lesquelles une transformation dégradante est acceptée. Son fonctionnement est le suivant : initialement la température T est fixée à valeur donnée T_0, puis une solution S est générée à l'aide d'une heuristique quelconque, ainsi S devient la solution courante. À chaque

itération une solution S' est choisie dans un voisinage de S, si S' est meilleure que S alors S' devient la solution courante sinon S' devient la solution courante avec une probabilité dépendant de T et de la différence de coût entre S et S'. Plus la température est haute, plus la probabilité d'accepter une transformation qui dégrade la solution courante est élevée. Au fur et à mesure des itérations, la température diminue de sorte qu'il devient de moins en moins probable d'accepter une solution dégradante. L'efficacité de cet algorithme dépend bien sûr, entre autre, de la stratégie adoptée pour faire décroître T. Différents mécanismes de contrôle sont proposés dans [12].

Algorithmes évolutionnaires

Les algorithmes évolutionnaires sont inspirés du domaine de la biologie : ils ont été introduits dans les années 1950 [50]. Les techniques utilisées reproduisent le schéma d'évolution des espèces et en adoptent le vocabulaire. Les solutions sont appelées individus et l'algorithme traite simultanément plusieurs individus. L'ensemble de ces individus est appelé population et évolue à chaque itération de l'algorithme. La population relative à une itération donnée s'appelle génération, on obtient donc une nouvelle génération à chaque itération. Les individus qui servent à produire la nouvelle génération sont appelés parents et les individus résultants sont appelés enfants ou fils. Le principe des algorithmes évolutionnaires est simple : l'idée est de faire évoluer une population initiale grâce à des mécanismes de reproduction et de sélection de manière à obtenir des individus de plus en plus performants. Dans le cadre des problèmes de minimisation un individu S est d'autant plus performant que son coût ($f(S)$) est faible. Le schéma général de cet algorithme fonctionne comme suit. Initialement, on génère un nombre aléatoire d'individus afin de construire la population initiale. A chaque itération on choisit des individus pour la reproduction et on leur applique des opérateurs de croisement et de mutation. Les opérateurs de croisement (souvent désignés par leur équivalent anglais *crossover*) produisent un ou deux enfants à partir de deux parents tandis que les opérateurs de mutation produisent un enfant à partir d'un unique individu. La performance des nouveaux individus est évaluée et un opérateur de sélection choisit les individus qui appartiendront à la prochaine génération. Le mécanisme de sélection permet de garder une population de taille constante et favorise la survie des individus les plus performants. La fonction de *Selection_Reproduction* sélectionne une portion d'individus de la population courante sur laquelle les opérateurs d'évolution seront appliqués, la fonction de *Croisement* applique des opérateurs de croisement sur les individus sélectionnés, la fonction de *Mutation(P)* applique des opérateurs de mutation sur les individus, et la fonction de *Selection* choisit un ensemble d'individus afin de les garder pour former la population de la génération suivante.

Plusieurs approches d'algorithmes évolutionnaires [128, 48] ont été étudiées :

- Les stratégies d'évolution (*evolution strategies* - ES), utilisées pour résoudre des problèmes d'optimisation continue.
- La programmation évolutionnaire (*evolutionary programming* - EP), conçue pour faire évoluer des automates à états finis, l'idée étant de créer une intelligence artificielle.
- Les algorithmes génétiques (*genetic algorithms* - GA), utilisés pour résoudre des problèmes d'optimisation combinatoire, ces derniers étant certainement les plus populaires.

À l'origine, ces trois approches ont été développées en parallèle s'ignorant mutuellement. Aujourd'hui les algorithmes évolutionnaires sont massivement utilisés, ce qui conduit à des techniques de plus en plus sophistiquées ainsi qu'à de nombreuses variantes. Pour plus d'information sur ces techniques on peut se référer à [10] en particulier aux deux premiers chapitres présentant de manière générale les algorithmes évolutionnaires, puis détaillant leurs différentes caractéristiques ainsi que la manière de les implémenter.

Les algorithmes évolutionnaires manipulent une population de solutions qu'ils font évoluer grâce des mécanismes basés uniquement sur des opérateurs de croisement, mutation et sélection. Au cours des années, il est devenu classique d'ajouter une recherche locale avant la phase de sélection des nouveaux individus, rendant l'algorithme bien plus efficace et créant ainsi un type d'algorithme hybride "algorithme évolutionnaire / recherche locale". Ces algorithmes hybrides sont connus sous le nom d'algorithmes *mémétiques*.

Colonie de fourmis

Les algorithmes de colonies de fourmis (approche proposée par Colorni et al. [33]) s'inspirent du comportement de certains insectes sociaux, en particulier des fourmis. Ces dernières parviennent collectivement à résoudre des problèmes trop complexes pour un unique individu. Afin de comprendre le mécanisme de ces algorithmes, il faut tout d'abord s'intéresser à la manière dont les fourmis résolvent le problème consistant à trouver le plus court chemin entre leur nid et une source de nourriture. Lorsqu'elles sont à la recherche de nourriture, les fourmis explorent l'espace autour de leur nid de manière aléatoire. En se déplaçant, elles laissent des phéromones sur le sol. Lorsqu'une fourmi trouve une source de nourriture, elle retourne à son nid. Les fourmis qui ont emprunté le plus court chemin arrive plus tôt à la source de nourriture, elles reprennent donc le même chemin pour le retour avec une forte probabilité renforçant ainsi la présence de phéromones sur ce trajet. Une fourmi qui cherche son chemin aura tendance à suivre une piste déjà marquée par des phéromones. La plus forte présence de phéromones sur le plus court chemin encourage les autres fourmis à suivre ce trajet pour se rendre à la source de nourriture. A force d'allers-retours, ce chemin est de plus en plus marqué et sera donc suivi à terme par l'ensemble de la colonie. Il existe plusieurs schémas algorithmiques pour résoudre un problème d'optimisation par colonies de fourmis. Ces schémas sont assez complexes à formaliser de manière simple et compréhensible sous forme d'algorithme de principe. Cependant, leur point commun est d'inclure un modèle de phéromones qui sert à guider la recherche de solutions vers les régions plus prometteuses de l'espace de recherche. Ils fonctionnent en deux étapes :

- Étape 1 : des solutions sont choisies dans l'espace de recherche en utilisant un modèle donné de phéromones
- Étape 2 : les solutions générées sont évaluées et leur coût sert à modifier le modèle de phéromones de manière à concentrer la recherche de solutions dans les régions contenant des solutions de bonne qualité.

Des informations bien plus détaillées sur ces techniques sont étudiées dans [19]. Comprenant entre autre le modèle biologique à l'origine des algorithmes de colonies de fourmis, le schéma général de résolution par colonies de fourmis et les variantes algorithmiques les plus répandues.

2.2.4 Programmation linéaire

La programmation linéaire est un outil classique de recherche opérationnelle qui peut s'adapter à un grand nombre de problèmes différents. On appelle programme (ou modèle) linéaire un modèle mathématique représentant un problème d'optimisation, dans lequel le but est d'optimiser une fonction objectif linéaire sous un certain nombre de contraintes. Ces contraintes sont modélisées par des équations ou inéquations linéaires. Il existe des méthodes qui assurent la résolution exacte d'un tel programme. Une des méthodes les plus connues est la méthode du simplex [38] (de son inventeur G.B. Dantzig) qui a en théorie une complexité non polynomiale. Cependant, il s'avère qu'elle est très efficace en pratique. Comme toutes les techniques de résolution exacte, les temps de calcul peuvent vite devenir très grands, il est courant d'utiliser des techniques supplémentaires comme la génération de colonnes ou le *branch and cut* pour les problèmes de grande taille. Des librairies d'optimisation comme IBM ILOG CPLEX [2] [35] implémentent ces méthodes. Le terme programmation linéaire suppose que les solutions à trouver doivent être représentées par des variables réelles. Il existe aussi la programmation linéaire en nombres entiers dans laquelle les variables sont entières et la programmation linéaire mixte dans laquelle les variables peuvent être entières ou réelles. La résolution d'un programme linéaire en nombres entiers (ou mixtes) est nettement plus difficile que la résolution d'un programme linéaire à variables réelles (à cause des contraintes d'intégrité supplémentaires). D'autre part, il existe généralement plusieurs modélisations possibles par programmation linéaire d'un problème et il est très difficile de déterminer a priori le meilleur modèle.

2.3 Gestion de la consommation énergétique en environnements virtualisés

Cette section présente les différentes solutions pouvant être utilisées dans la gestion d'un *data center* afin d'améliorer son rendement énergétique. Aussi, une présentation de travaux de recherche intégrant ces techniques est proposée.

2.3.1 Outils de gestion de consommation énergétique

Les outils pouvant permettre de réduire la consommation énergétique d'une ou d'un ensemble de machines physiques sont des solutions qui s'appliquent à différents niveaux (machine physique, machine virtuelle, réseau, ...). Toutes ces solutions ont pour même but de réduire globalement la consommation d'un *data center*. En effet, chaque amélioration (diminution de la consommation) de chacune des machines physiques composant un *data center* amène au final à une réduction de la consommation globale non négligeable étant donné le grand nombre de machines physiques qui compose un *data center*.

- La première solution concerne l'extinction de machines physiques lorsque celles-ci sont trop peu utilisées ou pas du tout utilisées, rendant leur rendement très médiocre. Bien que très simple et intuitive, cette méthode requiert un processus de déplacement des machines virtuelles s'exécutant sur la ou les machines physiques concernées afin de pouvoir les éteindre sans détruire les services

2. http ://www-01.ibm.com/software/in/integration/optimization/cplex/

qu'elles hébergeaient. Le procédé inverse : ré-allumage d'une ou de plusieurs machines physiques est évidemment également utilisé si toutes les machines physiques déjà allumées sont surchargées et que de nouvelles machines virtuelles doivent être allouées.

- Comme évoqué ci-dessus, il est souvent nécessaire de pouvoir déplacer des machines virtuelles entre différentes machines physiques. Ce processus est appelé "migrations de machines virtuelles" [72] permet de déplacer une machine virtuelle, et tout son environnement (données stockées en mémoire), d'une machine physique à une autre en fonction des terminaisons d'exécutions d'autres machines virtuelles. Ces transferts ne sont pas sans frais pour la consommation énergétique car ils nécessitent un certain temps durant lequel les machines virtuelles en migration consomment des ressources sur les machines physiques source et destination. Il est donc très important de le faire à bon escient pour éviter des surcoûts inutiles. Le fait de déplacer des machines virtuelles permet deux choses :

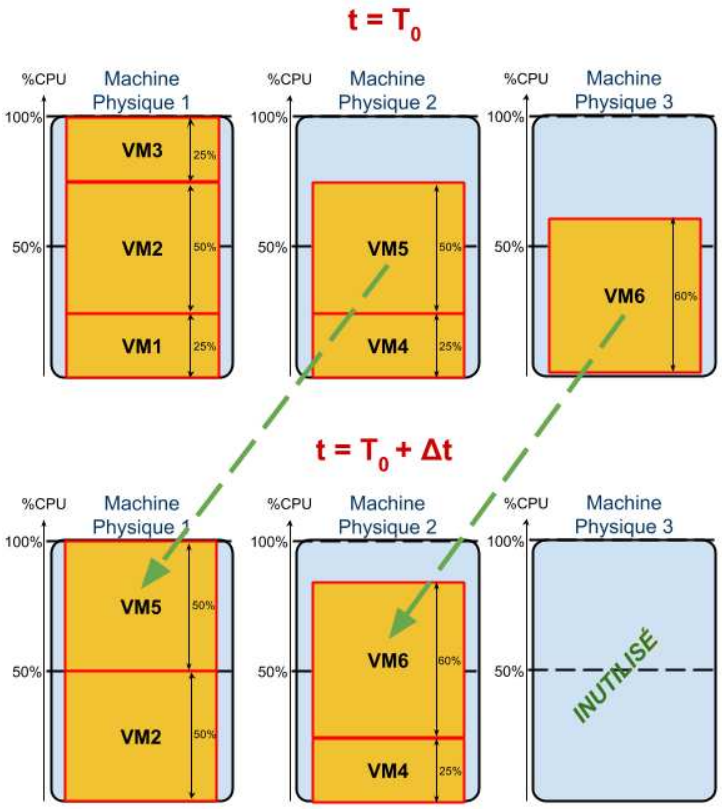


Figure 2.10 – Illustration de l'utilisation du processus de migration de machines virtuelles après terminaison d'exécution de certaines d'entre elles (les machines 1 et 3 dans cet exemple), afin de les consolider dans un minimum de machines physiques.

- Libérer de la place sur une ou des machines physiques et ainsi les éteindre complètement lorsque celles-ci sont vides

- À l'inverse, rassembler le plus possible de machines virtuelles sur une machine physique (technique de consolidation) pour utiliser au maximum ses capacités et ainsi améliorer son rendement.

Dans l'exemple illustré par la Figure 2.10, 6 machines virtuelles et 3 machines physiques sont considérées. Dans cet exemple simpliste, en considérant que les machines physiques sont homogènes en termes de capacité CPU et que le placement des machines virtuelles n'est pas contraint en termes de capacité Mémoire, alors les six machines virtuelles occupent entre 25 et 60% des machines physiques sur lesquelles elles sont affectées. À $t = T_0$, les six machines virtuelles sont en cours d'exécution, ce qui donne un taux d'utilisation CPU des machines physiques de :

- 100% pour la machine physique 1
- 75% pour la machine physique 2
- 60% pour la machine physique 3

En considérant qu'une reconfiguration du système visant à diminuer le nombre de machines physiques (et donc potentiellement la consommation énergétique) a lieu à $t = T_0 + \Delta t$, certaines machines virtuelles ont pu, durant l'intervalle de temps $[T_0; T_0 + \Delta t]$ terminer leur exécution. Si l'on considère, qu'aucune nouvelle machine virtuelle n'est apparue et que les machines virtuelles 1 et 3 ont fini leur exécution durant cet intervalle, alors le nombre de machines virtuelles encore en vie a diminué. Ainsi, une ré-allocation des machines virtuelles sur les machines physiques peut être appliquée. La machine virtuelle 1 est attribuée à la machine physique 5, puis la machine virtuelle 6 est allouée sur la machine physique 2. Avec ce nouveau placement, la machine physique 1 est toujours utilisée à 100% de ses capacités, la machine physique 2 est utilisée à 85%, et la machine physique 3 devient totalement vide. Ce nouveau placement permet donc de continuer les exécutions des machines virtuelles toujours en vie, et d'éteindre la machine physique 3 désormais inutilisée.
De nombreuses études ont été menées dans le domaine des migrations de machines virtuelles, analysant et expliquant notamment leur fonctionnement mais étudiant également dans quelles mesures leur utilisation est bénéfique dans les approches et les algorithmes visant à réduire la consommation énergétique des *data center* : [132] [119] [133] [89] [71]

- Enfin, le DVFS (*Dynamic Voltage and Frequency Scaling*) [79, 62] permet de changer dynamiquement la fréquence des CPUs des machines physiques en fonction de leurs taux d'utilisation. Cet outil, implémenté dans le noyau Linux depuis plus de 10 ans est détaillé dans la Section 2.3.1 et a fait l'objet d'une des contributions de cette thèse par son intégration dans le simulateur CloudSim.
  L'idée clé de la mise en œuvre du DVFS est de réduire la fréquence (GHz) du CPU et de la tension (V) pendant des périodes de faible utilisation du processeur. Une réduction de la fréquence et de la tension engendre inévitablement une diminution de la puissance délivrée par CPU en raison de la nature des circuits CMOS (*Complementary Metal Oxide Semiconductor*) d'aujourd'hui [108]. En particulier, la puissance dissipée par les circuits CMOS [4] est composée de deux parties : statique et dynamique. La partie statique est entre autre du aux courants de fuite du composant, dont la diminution est possible lors de la conception des circuits CMOS. D'autre part, la partie dominante,

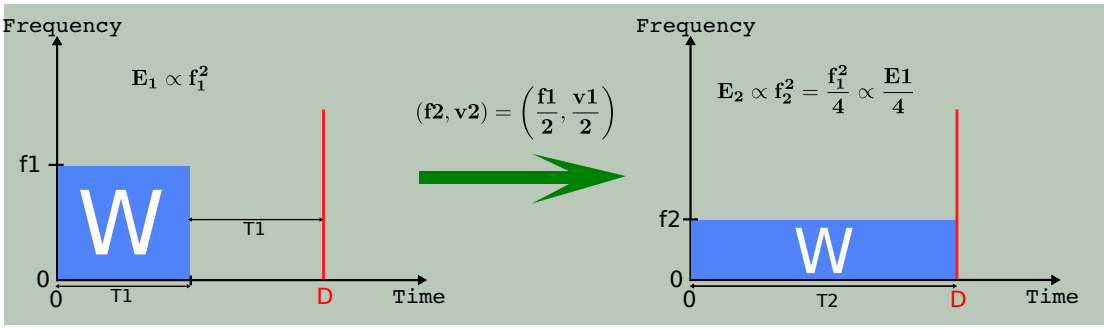


Figure 2.11 – Illustration du gain énergétique à l'utilisation d'une fréquence plus faible au détriment de la durée d'exécution de celle-ci.

dite dynamique, due à la charge et la décharge des composants du CMOS. Cette consommation dynamique est communément approximée par l'équation suivante [66] :

$$P = C \times f \times V^2 \tag{2.3.1}$$

où C est la capacité de commutation ("*switching factor*" en anglais), f la fréquence de commutation, et V la tension d'alimentation. Selon cette équation, une réduction linéaire de tension V implique une réduction quadratique de puissance fournie par le composant. Cependant, une diminution de la tension V implique également une réduction de la vitesse de commutation des transistors composant le CMOS, ce qui conduit inévitablement à une baisse de la fréquence maximale théoriquement possible du processeur. Par conséquent, pour assurer le bon fonctionnement du CPU, sa fréquence doit être abaissée proportionnellement à la baisse de tension appliquée. C'est ainsi que l'utilisation du DVFS est associée à des couples, appelés *P-states* et *C-states* : les *P-states* sont associés à des états actifs du CPU (exemple en Tableau 2.1), alors que les *C-states* sont associés à des états inactifs (*"Idle"* en anglais) du CPU [3] [4]. Il est également à noter qu'un abaissement de la tension de fonctionnement du CPU diminue également les courants de fuite des transistors, participant aussi également ainsi au gain de puissance consommée. Le temps de transition, très rapide, entre ces différents couples de fonctionnement fréquence/voltage est de l'ordre de la dizaine de microsecondes [21].

P-state	Fréquence	Voltage	Puissance
P0	1.6 GHz	1.484 V	∼25 Watts
P1	1.4 GHz	1.420 V	∼17 Watts
P2	1.2 GHz	1.276 V	∼13 Watts
P3	1.0 GHz	1.164 V	∼10 Watts
P4	800 MHz	1.036 V	∼8 Watts
P5	600 MHz	0.956 V	∼6 Watts

Table 2.1 – Présentation des *P-states* et leur couple fréquence/voltage ainsi que leur puissance associée, pour le processeur Intel Pentium M 1.6GHz [34].

3. https ://software.intel.com/en-us/blogs/2008/03/12/c-states-and-p-states-are-very-different
4. https ://software.intel.com/en-us/articles/power-management-states-p-states-c-states-and-package-c-states

Ainsi, l'utilisation de ces différents couples tension/fréquence des CPUs, en fonction de la capacité de traitement nécessaire à un instant t, diminue la rapidité de calcul du CPU comme illustré en Figure 2.11 (diminution de sa fréquence f de fonctionnement), mais permet ainsi de réduire la puissance (W) qu'il délivre (abaissement de sa tension V de fonctionnement). L'intérêt de l'utilisation du DVFS réside donc dans le fait d'utiliser ses changements de fréquence CPU à bon escient. Ainsi, de multiples études ont déjà été menées dans ce domaine. Certaines s'intéressent à trouver, pour des applications découpées en différentes phases d'utilisation des ressources, une fréquence optimale à adopter pour les phases d'utilisation du CPU. Cette approche est utilisée en se basant sur des traces existantes d'intervalles d'utilisation CPU ou de données, et utilisant ainsi des algorithmes de prédiction d'utilisation. Ce sont donc ces premières études [143, 58] qui ont utilisées le DVFS avec une approche de prédiction de charge CPU essayant ainsi d'utiliser la fréquence optimale à chaque instant.
D'autres approches se basent sur l'exécution de tâches dont les dates de fin d'exécution sont connues à l'avance. Ces approches plus simplistes, permettent donc de déduire les fréquences optimales d'exécution de chaque tâche afin de respecter au mieux leur date de fin d'exécution [150, 109]. Bien sûr, ces approches permettent d'obtenir un excellent compromis entre la performance (temps d'exécution) et la réduction de consommation d'énergie obtenue [46].

Afin d'illustrer les économies que rende possible l'utilisation du DVFS, voici un exemple :

Exemple 2.1 : Économie d'énergie théorique

- Soit une machine physique h, et F_1 et F_2 deux de ses fréquences, avec $F_2 = \frac{F_1}{2}$
- Soit F_1 la fréquence de départ de 2GHz et F_2 égale à $\frac{F_1}{2}$ soit 1GHz
- Soit la tâche W : exécutée à la fréquence F_1 son temps d'exécution est égal à $T_1 = 900$ secondes, exécutée à la fréquence F_2 sont temps d'exécution est donc de $T_2 = 1800$ secondes

En prenant comme base la configuration présentée en Figure 2.11, il faut prendre en compte l'hypothèse forte présentée dans cet exemple admettant que les différentes fréquences d'un CPU sont proportionnelles aux voltages de fonctionnement. Alors dans ce cas la relation quadratique (Équation 2.3.1) entre la puissance et la fréquence de fonctionnement est directe :

- Soit la fréquence $F_1 = 2$GHz, $V_1 = 2$v, alors P_1 délivrée par h à 100% d'utilisation CPU égale 8 watts
- Soit la fréquence $F_2 = 1$GHz, $V_1 = 1$v, alors P_1 délivrée par h à 100% d'utilisation CPU égale 1 watt

alors, dans ces conditions, l'exécution de W consommera :

- À la fréquence F_1 : $\frac{8\times900}{3600} = 2Wh$
- À la fréquence F_2 : $\frac{1\times1800}{3600} = 0.5Wh$

Cela dit, comme on peut le remarquer dans le Tableau 2.1 la fréquence n'est pas forcément, selon les processeurs, linéaire par rapport au voltage de fonctionnement et la puissance n'est pas donc pas toujours proportionnelle au carré de la fréquence.

Exemple 2.2 : Économie d'énergie pour le processeur Intel Pentium M

- Soit la fréquence $F_1 = 1.6$GHz, et P_1 délivrée par h à 100% d'utilisation CPU égale 25 watts (correspondant à PO dans le tableau)
- Soit la fréquence $F_2 = 0.8$GHz, et P_2 délivrée par h à 100% d'utilisation CPU égale 8 watts (correspondant à PO dans le tableau)

Dans ce cas, avec des données réelles l'économie d'énergie pour la même tâche W est de :
- À la fréquence F_1 : $\frac{25\times900}{3600} = 6.25Wh$
- À la fréquence F_2 : $\frac{8\times1800}{3600} = 4Wh$

Soit une diminution de 36% ce qui est loin de correspondre au 400% de réduction obtenus en considérant que la fréquence de fonctionnement de d'un CPU est proportionnelle au voltage nécessaire pour le faire fonctionner.

DVFS dans le noyau Linux

Le DVFS est implémenté dans le noyau Linux depuis 2001 et peut être utilisé en installant le paquet nommé *cpufrequtils*. Une documentation, accessible dans le dossier (*linux-x.y.z/Documentation/cpu-freq*) des sources du noyau explique de façon très précise le comportement de chacun de ses modes de fonctionnement. Le code source de l'implémentation du DVFS dans le noyau est également disponible dans le dossier source : *linux-x.y.z/drivers/cpufreq*.

Ce module DVFS permet au système de fonctionner dans 5 modes différents, selon les besoins de l'utilisateur, en sélectionnant le *gouverneur* correspondant au mode voulu. Les trois premiers sont des modes dits "statiques" car ils n'impliquent aucun changement de fréquence au cours du temps, contrairement aux deux derniers qui sont dits "dynamiques" ce qui signifie que leur *gouverneur* est capable de prendre une décision de changement de fréquence au cours de leur utilisation.

Voici la description détaillée du comportement de chacun de ces cinq modes [5] :

- *PowerSave* : ce mode utilise uniquement la fréquence la plus basse admise par le CPU de la machine. Quelle que soit l'évolution de la charge CPU, aucun changement de fréquence ne sera effectué par la *gouverneur* de ce mode.

- *Performamnce* : ce mode utilise uniquement la fréquence la plus haute admise par le CPU de la machine. Quelle que soit l'évolution de la charge CPU, aucun changement de fréquence ne sera effectué par la *gouverneur* de ce mode.

- *UserSpace* : ce mode utilise uniquement la fréquence qui a été choisie par l'utilisateur, parmi les fréquences admises par le CPU de la machine. Une fois une fréquence choisie, aucun changement de fréquence ne sera effectué par la *gouverneur* de ce mode quelle que soit l'évolution de la charge CPU. Seul un changement de fréquence effectué par l'utilisateur lui même sera pris en compte.

5. https ://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

- *Conservative* : ce mode dynamique fonctionne avec deux seuils (inférieur et supérieur). C'est à dire qu'il compare très régulièrement la valeur de la charge CPU à ces seuils. Si la charge CPU est inférieur au seuil inférieur pré-défini, alors la fréquence est abaissée à la fréquence inférieure la plus proche. Inversement, si la charge CPU est supérieure au seuil supérieur, alors la fréquence est augmentée à la fréquence supérieure la plus proche (si la fréquence la plus haute est déjà en cours d'utilisation, cela n'implique aucun changement). Ce mode a donc un comportement très souple en appliquant des changements de fréquence pas à pas.

- *OnDemand* : ce mode dynamique fonctionne avec un seul seuil, par rapport auquel il compare la valeur de la charge CPU. Si la valeur de la charge CPU est supérieure à ce seuil, alors la fréquence la plus haute est directement affectée. Une diminution de la fréquence est appliquée lorsque que le gouverneur détecte que la charge CPU est inférieure à ce seuil depuis un certain nombre de comparaisons (paramètre de décision réglable). Alors la fréquence est abaissée à la fréquence inférieure la plus proche. Ce mode est beaucoup plus réactif que le mode *Conservative* en affectant la fréquence la plus haute dès que la charge CPU dépasse le seuil de comparaison. Cela lui permet d'être très réactif à des changements brusques de charge tout en abaissant la fréquence lors des phases de faible utilisation.

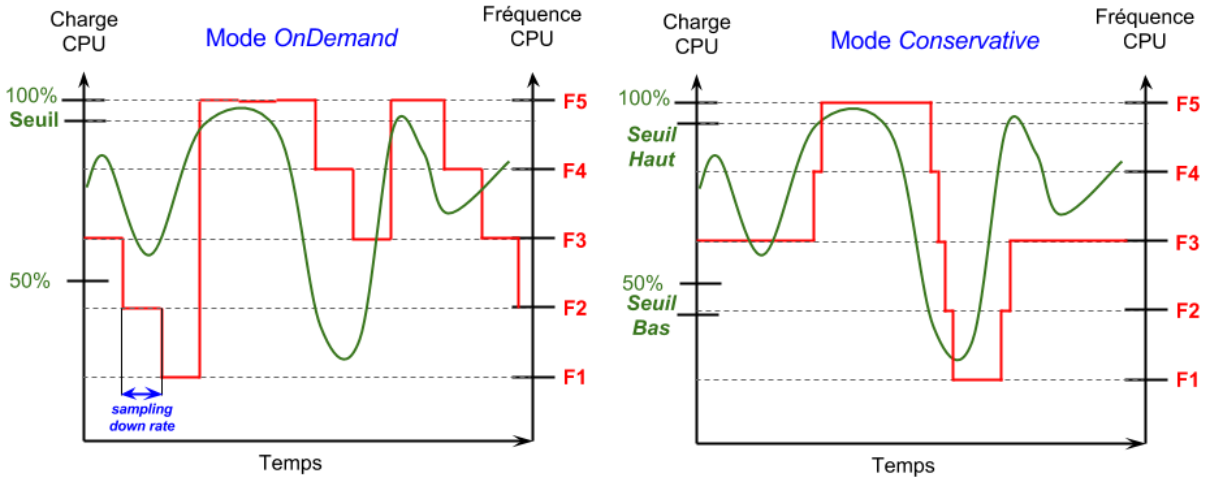


FIGURE 2.12 – Exemples simplistes des comportements de changements de fréquence des modes *OnDemand* et *Conservative* en fonction de la charge CPU.

Exemple 2.3 : Comportement des modes dynamiques : OnDemand et Conservative

Un exemple très simple des comportements respectifs du mode *OnDemand* et du mode *Conservative* est illustré en ficgure 2.12. Sur cet exemple la courbe de charge CPU est la même sur les deux graphiques, mais cela ne signifie pas que le nombre d'instructions traitées par le CPU est le même. En effet, dès qu'un changement de fréquence est appliquée, la charge CPU varie proportionnellement à ce changement de fréquence. Dans cet exemple, les changements de fréquence n'étant pas identiques, dû aux deux compor-

tements différents des deux modes, les courbes de charge CPU identiques permet uniquement d'illustrer à quels moment et comment les prises de décisions de ces gouverneurs sont faits.

Une fois le paquet activant le DVFS installé et configuré, un certain nombre d'informations concernant le système et les *gouverneurs* sont disponibles :

- CPU(s) affectés par le DVFS
- Les *gouverneurs* disponibles (un pour chaque mode)
- Les fréquences disponibles sur le ou les CPU de la machine. Ces fréquences sont directement liées à la génération et au modèle de CPU intégré à la machine
- Les fréquences maximum et minimum sont également enregistrées dans des fichiers séparés. Dans certain cas, il peut être utile de pouvoir affecter des valeurs spécifiques à ces fréquences.
- *Transition latency* : définit l'intervalle minimum entre deux mesures de la charge CPU par le système
- Le mode (*governeur*) actuellement utilisé

Chaque mode DVFS peut être configuré de façons différentes en spécifiant certaines valeurs de paramètres :

- *Sampling rate* : définit l'intervalle minimum entre deux changements de fréquence par le système
- *Thresholds* : Les modes *OnDemand* et *Conservative* adaptent la fréquence CPU à utiliser en effectuant une comparaison de la charge CPU à des seuils pré-définis. En spécifiant des valeurs personnalisées pour ces seuils de décisions, l'utilisateur peut donc adapter leur comportement.
- *Sampling down factor* : En mode *OnDemand*, ce paramètre défini le nombre de fois que le système compare la valeur de charge CPU à la valeur du seuil avant de prendre la décision de baisser la fréquence.

D'autres informations à propos des statistiques d'utilisation de chacun des modes sont disponibles :

- Temps d'utilisation passé dans chacune des fréquences du CPU
- Le nombre total des changements de fréquences effectués.
- Un tableau récapitulatif des changements de fréquence affichant le nombre de transitions effectuées entre chacune d'entre elles.

Remarque 1 : Effet sur les E/S :

Un abaissement de fréquence, qui induit l'utilisation d'un voltage plus faible permettant de réduire la consommation d'énergie, provoque irrémédiablement un ralentissement de la vitesse de calcul du CPU. Ce changement de fréquence CPU est réalisé en changeant la valeur du multiplicateur entre la fréquence du FSB (*Front Side Bus*) et du CPU. Outre ce fonctionnement normal, sur certaine architecture de carte mère, ce changement de voltage peut également affecter d'autres composants, par exemple : la vitesse des E/S (Écriture/Lecture sur le disque dur). Cela arrive quand le changement de fréquence ne change pas uniquement la valeur du multiplicateur, mais directement la fréquence FSB. Ce qui entraîne donc un

ralentissement de tous les composant qui se basent sur cette fréquence.

Remarque 2 : Changement de fréquence commun entre cœurs de CPU :

Les changements de fréquences des cœurs des CPUs dépendent directement de l'architecture du CPU. Par exemple, il se peut qu'un changement de fréquence sur un cœur affecte également un autre cœur, voir tous les cœurs du CPU. Afin de mettre en évidence ces différents comportement lors des changements des fréquences des cœurs des CPU, des expériences ont été menées sur la plate-forme RECS de l'IRIT. Le détail et les résultats de ces expérimentations sont exposés en Annexe A.

2.3.2 Études énergétique et de QoS pour le Cloud Computing

Abdelsalam et al. [1] proposent une analyse mathématique entre les SLA et le nombre de serveurs utilisés. Dans leur modèle, l'aspect énergétique est présent avec la prise en compte du DVFS appliqué à l'ensemble des machines physiques, considérées comme hétérogènes, permettant ainsi chacune d'entre elles d'utiliser des fréquences de fonctionnement CPU différentes. Leur étude tient également compte du nombre d'utilisateurs et de leurs besoins, ainsi que du temps de réponse moyen résultant du nombre de requêtes envoyées aux serveurs.

Islam et al. [67] présentent une modèle d'élasticité pour les *Clouds*. Chaque type de ressources (CPU, Mémoire, Bande passante réseau, etc...) peut être alloué de façon très fine, et les utilisateurs ont conscience des ressources allouées, ainsi que des métriques de Qualité de Service intéressantes pour l'exécution de leur requêtes (comme dans le cas de Amazon CloudWatch [6]). Leur modèle intègre le coût d'approvisionnment des ressources sous-utilisées, et le coût de la dégradation de performance dû à ces sous-approvisionnements de ressources. Le sur-approvisionnement est la différence entre la capacité de traitement disponible et la capacité réelle que nécessite les requêtes à traiter, alors que le sous-approvisionnement est estimé par le pourcentage de requêtes qui ont dû être rejetées.

Gelenbe et al. [54] ont formulé par un problème d'optimisation la répartition de charge entre un service de *Cloud Computing* local et distant. Leur étude définit une fonction multi-objectifs afin d'optimiser le temps de réponse et la consommation énergétique par tâche. Pour cela ils utilisent un taux d'arrivée de tâche suivant une loi de Poisson.

Baliga et al. [14] proposent une étude de la consommation d'énergie pour le traitement et la gestion de très grandes quantités de données. Leur modèle associe trois types de services *Cloud* : *Storage as a Service*, *Software as a Service*, et *Processing as a Service*. La prise en compte du coût énergétique a été intégrée à la chaîne logistique de problèmes incluant l'exécution, le stockage et le transport des données sur le réseau, ceci pour les *Clouds* privés et publiques.

Garg et al. [52] ont utilisé un modèle énergétique tenant compte de plusieurs caractéristiques : consommation énergétique, taux d'émission de gaz carbonique, charge des serveurs, et le rendement des CPU. Leur modèle se compose de CPU homogènes, un système de refroidissement qui dépend d'un coefficient de performance (COP), et inclut également l'utilisation du DVFS. Leur métrique d'évaluation de performance prend en compte : les moyennes de consommation d'énergie et d'émission de carbone, des classes d'applications plus ou moins urgentes, un coût de transfert de données et les profits engendrés.

6. http://aws.amazon.com/fr/cloudwatch/

Beloglazov et al. [17] ont étudié la dégradation des performances des machines virtuelles en utilisant un modèle tenant compte : du CPU, de la mémoire et de la bande passante réseau utilisée. Leur modèle intègre un paramètre de Qualité de Service qui est l'approvisionnement (en poucentage CPU) des machines virtuelles. Ainsi, une violation de SLA surgit si une machine virtuelle ne reçoit pas le pourcentage de CPU requis dans le SLA. Dans [16] ils proposent une technique de gestion de ressources pour minimiser la consommation d'énergie, réduire les couûts d'exploitation des ressources et garantir un certain niveau de QoS. La technique de consolidation de machines virtuelles est utilisée pour diminuer les consommations d'énergie, et la Qualité de Service est interprétée par : un pourcentage de CPU et de Mémoire, ainsi qu'un certain débit réseau. Une violation de SLA est détectée si une machine virtuelle ne reçoit pas les caractéristiques de CPU, mémoire et débit réseau promis.

Mi et al. [96] ont formulé un problème d'optimisation multi-contraintes afin de trouver un placement qui optimise la consolidation des machines virtuelles afin de minimiser la consommation d'énergie. Leur approche utilise une prédiction de charge, et propose une heuristique basée sur les algorithmes génétiques afin de trouver une politique de reconfiguration efficace. La fonction objectif utilisée prend en compte la consommation énergétique et une métrique de pénalité si les pourcentages d'utilisation CPU n'est pas maintenu entre deux seuils pré-définis.

Duy et al. [41] propose une modélisation, une implémentation et une évaluation d'un algorithme de placement, intégrant un réseau de neurones, dédié à minimiser la consommation énergétique des serveurs d'un *Cloud*. Une prédiction de charge, basée sur un historique, est intégrée à leur algorithme afin que celui-ci puisse estimer s'il est intéressant d'éteindre ou d'allumer un certain nombre de serveurs.

Srikantaiah et al. [129] ont étudié comment obtenir une consolidation de machines virtuelles efficace basée sur les relations complexes qu'il peut y avoir entre : la consommation d'énergie, le taux d'utilisation des ressources et la performance. Leur étude montre qu'il existe un point de fonctionnement optimal entre tous ces paramètres, basé sur le problème de *Bin Packing*, et est appliqué au problème de consolidation de machines virtuelles.

Dans [39], Dasgupta et al. étudient la normalisation de *workload* dédiés au systèmes hétérogènes comme les *Clouds*. Les principales différences par rapport aux études précédentes dans ce domaine sont que leurs recherches se basent sur des *data center* hétérogènes et sur les avantages que donnent un système de contrôle du refroidissement pour des temps de traitements de tâches inconnus. Cette étude prend également en compte le ROI (*Return On investment*, et l'utilisation de la gestion dynamique des fréquences CPU (DVFS).

Les caractéristiques majeurs des travaux présentés dans ce manuscrit sont énumérées ci-dessous afin de positionner l'approche générale de cette thèse par rapport aux études cités ci-dessus :

- Définition des modèles d'architecture matérielle et logicielle
- Définition d'une modélisation de paramètres de QoS dédiés au *Cloud Computing* ainsi qu'un ensemble de métriques associées à ceux-ci
- Hétérogénéité en termes de puissance délivrée des machines physiques
- Utilisation du DVFS sur les machines physique
- Une gestion des ressources allouées aux machines virtuelles affectant les performances de celles-ci
- Une sélection de quatre métriques de QoS en relation directe avec des paramètres liés aux SLA

- L'utilisation d'algorithmes de placement, dont une méta-heuristique autorisant une analyse de l'influence d'une optimisation multi-objectifs appliquée à de métriques de QoS *Cloud*
- L'utilisation d'un simulateur de *Cloud*, dont les fonctionnalités ont été étendues en termes d'outils énergétique et d'évaluation de paramètres de QoS
- Analyses de l'impact de la modélisation de QoS, intégrée au sein d'une approche multi-critères, sur l'ordonnancement et la gestion du niveau global de QoS

L'ensemble des travaux de recherches de cette thèse ont à la fois des points communs avec plusieurs travaux cités ci-dessus, et ignorent également certains aspects. Une des différences importantes à souligner est que la notion de violation stricte de SLA n'est pas incluse pour plusieurs raisons : les cas de violations de SLA considérés dans les travaux cités ci-dessus (exemple : mauvais approvisionnement de capacité des machines virtuelles) ne correspondent pas à des réelles violations de clauses incluses dans les SLA de fournisseurs de services, mais le plus souvent à une dégradation de certain paramètres de performance du système. De plus, estimer à quel moment une violation apparaît, en tenant compte de paramètres inclus dans les SLA, implique un choix arbitraire de seuil de violation et donc une méthode d'évaluation peu convaincante. C'est pourquoi, une étude de l'évolution simultanée de plusieurs paramètres de QoS a été préférée.

2.4 Propositions de SLA de fournisseurs de services Clouds

De nos jours, le nombre de fournisseurs de services *Cloud Computing* ne cesse d'augmenter. Aussi, il est donc intéressant de comparer les contrats de SLA qu'ils proposent à leurs utilisateurs. D'une part pour analyser les paramètres pris en compte en fonction des types de services proposés, mais aussi afin d'avoir une vue d'ensemble de ces paramètres de QoS pour la suite des travaux menés lors de cette thèse. Le but de cette section n'est pas de donner et de comparer les définitions des paramètres de QoS des fournisseurs mais d'avoir une vue d'ensemble de la composition de ces SLA et ainsi pouvoir par la suite regrouper ces paramètres en catégories, puis les définir de manière plus précise.

Les sections suivantes présentent différentes catégories de définitions présentes dans les contrats de SLA des fournisseurs de services *Clouds* incontournables :

2.4.1 Amazon

Amazon définit SLA[7] pour leurs deux offres de Cloud Computing : Amazon EC2 (Elastic Compute Cloud) qui propose des services de calculs, et Amazon EBS (Amazon Elastic Block Store) qui fournit des services de stockage. Ils utilisent tous les deux les politiques SLA de Amazon Web Services Customer Agreement (the "AWS Agreement"). La dernière version a été mise à jour le 1^{er} Juin 2013 et elles sont appliquées séparément sur les comptes de Amazon EC2 et Amazon EBS.

Dans ce document Amazon définit les termes suivant : *Region Unavailable* et *Region Unavailability*. Cela signifie que certaines régions ne sont pas accessibles de l'endroit où les instances de l'utilisateur sont utilisées (*Availability Zone*). Les définitions de *Unavailable* et *Unavailability* données par Amazon sont les

7. http://aws.amazon.com/ec2/sla/

suivantes :

- Pour Amazon EC2, quand toutes les instances des utilisateurs n'ont plus de connexion avec l'extérieur.
- Pour Amazon EBS, quand tous les volumes de stockage des utilisateurs ne font aucune lecture ou écriture (E/S), alors qu'il y a des données en attente d'écriture ou de lecture (file d'attente non vide).

Une autre définition concerne ce que Amazon appelle le *Monthly Uptime Percentage* : il est calculé en soustrayant de 100% le pourcentage de minutes durant un mois pendant lesquelles Amazon EC2 ou Amazon EBS selon les cas, étaient en état de *Region Unavailable*.

Dans le document *AWE Customers agreement*, quelques aspects de la Qualité de Service sont définis :

- Changement de l'offre de service (*Service offering change*)
- Mise à jour des APIs (*APIs update*)
- Respect et sécurité des données (*Data security & privacy*)
- Disponibilité et impact des interruptions de services (*Availability and impact of temporary service suspension*)

2.4.2 Oracle

Oracle diffuse ses propositions de SLA dans un document appelé *Oracle Cloud Services Agreements*[8] disponible et téléchargeable pour beaucoup de pays différents. Tout d'abord, ce document contient une liste explicative des termes utilisés pour définir les règles de SLA qui sont proposées. Ce document est ensuite décomposé en plusieurs grandes catégories comme :

- Droits accordés (*Rights Granted*)
- Spécifications des services (*Service Specifications*)
- Utilisation des services (*Use of the Services*)
- Cycle de vie des services (*Services Period, End of Services*), etc ...

Outre ces catégories générales, des informations plus précises de paramètres de Qualité de Service peuvent être trouvées sur le site internet du : *Oracle Fusion Middleware Deployment Planning Guide for Oracle Directory Server Enterprise Edition*[9], qui propose également des services de type *Cloud Computing*. Par exemple, des paramètres définissant les qualités d'un système (*System Qualities)* sont identifiés en différentes catégories :

- Performance (*Performance*)
- Disponibilité (*Availability*)
- Extensibilité (*Scalability*)
- Sécurité (*Security*)

8. http://www.oracle.com/us/corporate/contracts/cloud-services/index.html
9. http://docs.oracle.com/cd/E29127_01/doc.111170/e28974/service-level-agreements.htm

- Dynamisme ou capacité latente (*Latent capacity*)
- Facilité d'utilisation *Serviceability*

2.4.3 Microsoft Azure

Microsoft Azure [10] diffuse leur SLA dans un document appelé *Windows Azure Cloud Services, Virtual Machines, and Virtual Network Service Level Agreement (SLA)*. Ce document contient des définitions de de termes contenus dans le SLA, et est composé de différentes catégories :

- Définitions de termes du SLA (*Definitions of terms*)
- Demandes de crédit de service (*Service Credit Claims*)
- Exclusions (*SLA exclusions*)
- Crédits de service (*Service Credits*)
- Niveaux de service (*Service Level*) : divisés en trois parties, cette catégorie contient des informations sur le paramètre de disponibilité par mois des services (*Monthly Connectivity Uptime Service Level*), et est défini à différents niveaux :
  - Services *Cloud* (*Cloud Services*)
  - Machines virtuelles (*Virtual Machines*)
  - Réseau virtuel (*Virtual Network*)

2.4.4 Google Apps

Les termes du contrat SLA proposé par Google Apps sont clairement exposés sur leur site internet [11]. Une version dédiée aux entreprises est également disponible [12].

Voici un résumé des SLA définis par Google :

- Interruption : Ce terme fait référence, pour un domaine, à un taux d'erreur utilisateur supérieur à cinq pour cent. L'interruption est mesurée sur la base du taux d'erreur constaté côté serveur.
- Services Google Apps couverts : Ce terme fait référence aux services concernés : *Gmail*, *Google Agenda*, *Google Talk*, *Google Documents & Drive*, *Google Groups*, *Google Sites & Google Apps Vault*. Les fonctionnalités *Labos* de *Gmail*,*Google Apps Postini Services* et les composantes de chat audio et vidéo de *Gmail* sont exclues du service.
- Pourcentage de disponibilité mensuelle : Ce pourcentage correspond, pour un mois calendaire donné, au nombre total de minutes de ce mois diminué de la durée (exprimée en minutes) des Interruptions, divisé par le nombre total de minutes du mois.
- Service : Ce terme désigne les services suivants fournis par Google au client dans le cadre du contrat : *Google Apps for Business* (également appelé Google Apps Édition Premier), *Google Apps for Government*, *Google Apps for ISPs* (également appelé Google Apps Édition Partenaire), *Google Apps for Education* (également appelé Google Apps Édition Éducation) ou *Google Apps Vault*.

10. http://www.windowsazure.com/fr-fr/support/legal/sla/
11. http://www.google.com/apps/intl/fr/terms/sla.html
12. http://www.google.fr/apps/intl/fr/terms/premier_terms.html

- Crédits de service : Un tableau explicatif sur les crédits de services est donné sur leur page internet de définitions de ces SLA. Google donne aussi des informations sur la manière dont le client doit agir pour obtenir ces Crédits de service, ainsi que leur condition d'application et leur limite maximale.

Contrairement aux paramètres de SLA décrits dans les propositions de SLA décrites ci-dessus, qui englobent des domaines nombreux et variés de la Qualité de Service, les études actuelles de modélisation ne proposent pas de définitions ou de métriques s'appliquant à ce genre de paramètres. Ce constat a amené à consacrer une large partie de cette thèse à une analyse des différents domaines de Qualité de Service intervenant dans le domaine du *Cloud Computing*, afin de pouvoir classer les paramètres et proposer des métriques mesurables et utilisables pour le maximum de paramètres.

2.5 Simulateurs de Cloud Computing

Plusieurs simulateurs de *Cloud Computing* sont actuellement en développement. En voici une liste non exhaustive, décrivant les caractéristiques de chacun d'entre eux.

- GroudSim [102] propose des outils de simulation événementiels pour les grilles de calcul et le *Cloud Computing*. Les principaux domaines de simulation sont les transferts de fichiers, la charge des ressources disponibles et le calcul des coûts de fonctionnement. Il peut être utilisé avec un *package* contenant des informations sur la probabilité de défaillance matérielle, le module d'événements peut alors détecter des erreurs se produisant sur la machine ou dans le réseau et lancer une procédure de reconfiguration sur les entités concernées.

- GreenCloud [76] fournit un environnement de simulation, implémenté en C++, dédié à l'analyse de la consommation d'énergie dans les centres de calcul. Il est principalement destiné aux études expérimentales sur les modes de communication et l'évaluation détaillée de la consommation d'énergie, des architectures actuelles mais aussi des futures architectures des *data centers*. GreenCloud est un simulateur au niveau paquet réseau, ce qui signifie qu'à chaque fois qu'un message doit être transmis, c'est une structure de paquets qui doit être allouée en mémoire puis exécutée. Bien que cette approche améliore significativement la précision de la simulation, cela augmente le temps global de simulation et conduit à des difficultés sur de très grandes instances (passage à l'échelle difficile). GreenCloud se focalise sur la modélisation détaillée des aspects de communication des données dans les réseaux des *data center*. Étant basé sur la plate-forme NS2 implémentant de bout en bout le protocole TCP/IP , cela permet de capturer la dynamicité des protocoles de communication tels que IP, TCP, UDP, etc. Ainsi, chaque message envoyé entre deux éléments, est fragmenté en un nombre de paquets dont la taille est limitée par le réseau MTU. Puis, lors de leur routage à l'intérieur du réseau du *Data center* ils peuvent être soumis à un certain nombres de problèmes typiques tels que des erreurs sur les liens ou de la congestion dans les routeurs réseau donnant lieu à des pertes de paquets. Pour être considéré comme correctement terminé, chaque objet du *workload* doit être correctement exécuté par les principaux composants de calcul et de communication) du simulateur. Le composant de calcul définit le nombre d'opérations qui doivent être exécutées avant une date limite de terminaison (dit *deadline*). L'inclusion de *deadlines* vise à introduire des contraintes de QoS spécifiées dans un SLA.

Le composant de communication détermine la quantité et la taille des des transferts de données qui doivent être effectués avant, pendant et après l'exécution du *workload*. Afin de couvrir la majorité des applications de *Cloud Computing*, GreenCloud définit trois types de *workload* : calcul intensif qui charge considérablement les serveurs de calcul, données intensives qui nécessitent des transferts de données très lourds, et un *workload* équilibré qui vise à modéliser les applications ayant des exigences à la fois de calcul et de transfert de données.

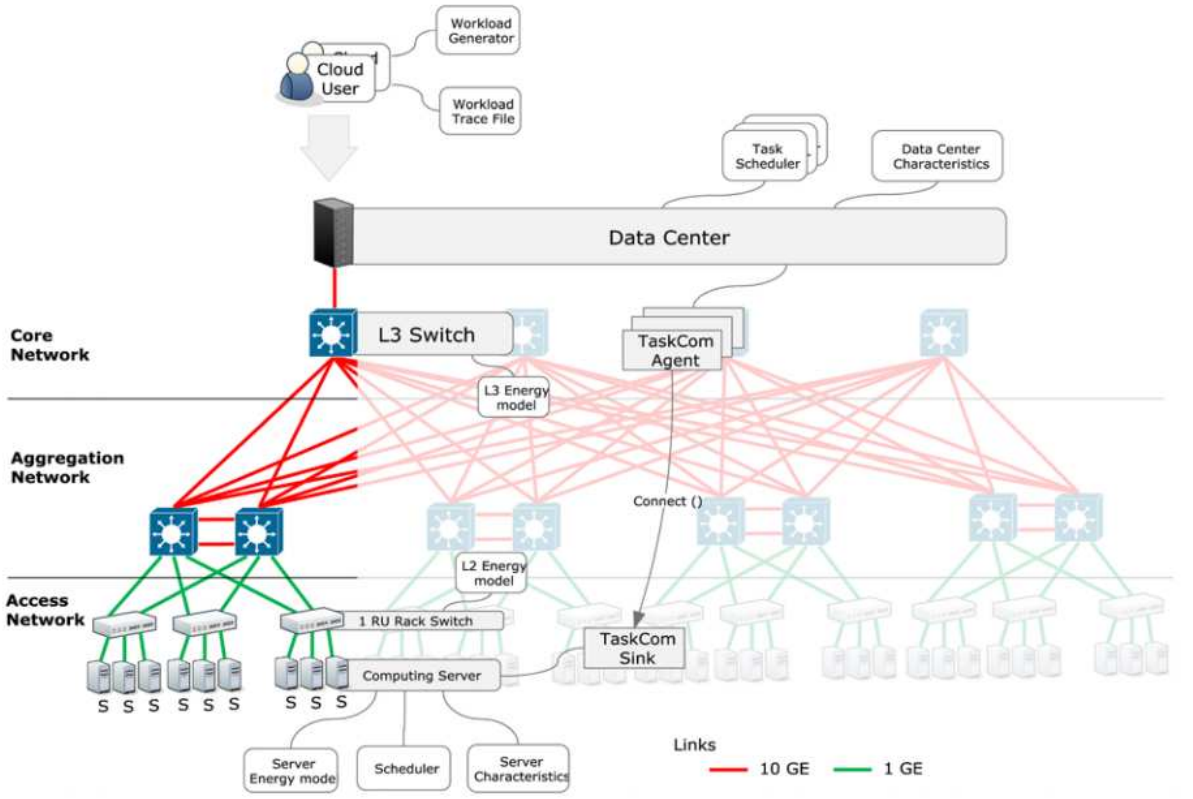


Figure 2.13 – Architecture de GreenCloud.

Les informations sur consommation d'énergie sont divisées en trois catégories : calcul, communication et infrastructure physique. Cette approche permet de modéliser la consommation d'énergie associée aux composants de calcul, aux switchs réseau ainsi qu'aux systèmes de refroidissement du *data center*. Les modèles énergétiques profitent ainsi de la granularité du simulateur (niveau paquet), ce qui permet de mettre à jour les niveaux de consommation d'énergie à chaque fois qu'un nouveau paquet arrive ou quitte un lien, ou à chaque fois qu'une tâche commence ou termine son exécution. GreenCloud implémente le DNS (Dynamic Network Shutdown) comme outil de réduction de consommation réseau, mais le DVFS n'est pas implémenté tel qu'il peut être utilisé dans le noyau Linux.

- Simgrid [26] est un projet conjoint entre l'Université d'Hawaii à Manoa, LIG Laboratoire de Grenoble et l'Université de Nancy. Il vise à fournir ensemble de fonctionnalités pour la simulation d'applications distribuées dans des environnements distribués hétérogènes. L'objectif principal du projet est de faciliter la recherche dans le domaine des systèmes à grande échelle, parallèles et distribués. En particulier, Simgrid fournit des environnements de programmation pour aider les chercheurs souhaitant analyser des algorithmes ou de réelles applications parallèles et qui ont donc besoin de pouvoir exécuter rapidement des simulations.

- DCWorms (*Data Center Workload and Resource Management Simulator*) [84], est un simulateur développé au laboratoire *Poznan Supercomputing and Networking Center* (PSNC) de Poznan en Pologne, basé sur l'ancien simulateur GSSIM [13] [83]. GSSIM avait été développé afin de proposer un outil automatisé pour les études expérimentales politiques d'allocation et d'ordonnancement de ressources dédiées aux systèmes distribués. DCworms est développé sur ces fonctionnalités de base de GSSIM, mais propose surtout des fonctionnalités supplémentaires liées aux études de consommation d'énergie et de rendement énergétique, devenus de réels enjeux dans les *data centers* d'aujourd'hui. DCWorms, dont l'architecture générale est décrite par la Figure 2.14, a donc pour principal objectif de permettre la modélisation et la simulation d'infrastructures de calcul afin d'estimer leurs performance, leurs consommation d'énergie, ainsi que différentes métriques de consommation d'énergie pour différents types de *workload* et différentes politiques d'ordonnancement.

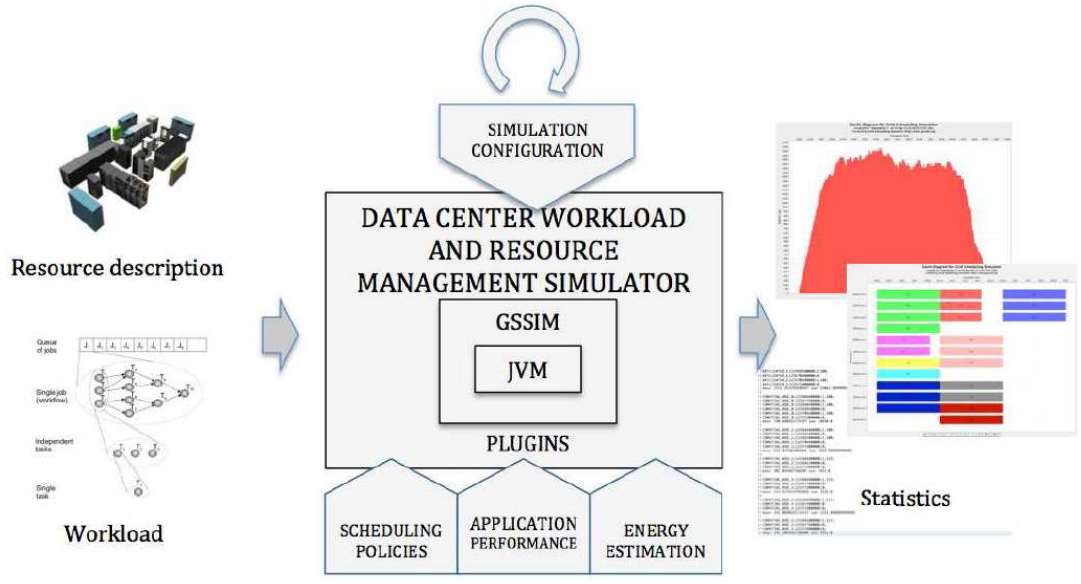


Figure 2.14 – Architecture de DCWorms.

Les données d'entrée du simulateur DCWorms sont constituées d'une description de l'architecture et des ressources de calcul, et d'une description du *workload* d'entrée. Ces données peuvent être directement fournies par l'utilisateur, DCWorms prend en charge les formats standard tel que SWF ou GWF, ou être générées en utilisant une fonctionnalité de GSSIM, appelé *workload generator*.

Cependant, les éléments clés de l'architecture de DCWorms sont des *plugins*. Ils permettent aux utilisateurs/chercheurs de configurer et d'adapter leur environnement de simulation en fonction des caractéristiques de leurs études : analyse de la performance, estimation de la consommation d'énergie, implémentation de politique de placement ou d'ordonnancement, etc... Chaque *plugin* peut être implémenté indépendamment et donc utilisé à la convenance de l'utilisateur en fonction de la spécificité de ses expérimentations. Les politiques introduites par l'utilisation de ces *plugins* sont alors appliquées sur l'ensemble des tâches en cours d'exécution et sur les ressources de calcul utilisées. Les résultats des expériences sont récupérés, sauvegardés, un rendu graphique est aussi proposé par un module de statistique permettant de visualiser les différentes mesures effectuées pendant la simulation. Les données de sorties des simulations sont donc composées de plusieurs statistiques permettant la comparaison des résultats des différentes politiques de placement et types de *workload* simulés. Son architecture modulable et l'ajout de *plugins* spécifiques permet donc à DCWorms de s'adapter à de nombreux problèmes de gestion de ressources mais également de répondre à des besoins différents entre chaque utilisateur. DCWorms permet également, d'un point de vue analyse énergétique, d'étudier la consommation des *data center* en incluant des modèles et des profils de consommation. En plus de pouvoir utiliser des modèles de puissance commun, pour l'analyse de la consommation des ressources de calcul, il est également possible d'intégrer des modèles tenant compte de la température des ressources et des systèmes de refroidissement d'un *data center*. Une description de l'approche par modèle thermique et de refroidissement est détaillée dans [18]

- CloudSim [24] et GridSim [22] sont deux outils de simulation événementiel séparés, implémentés en Java. Initialement, CloudSim a été développé sur la base de GridSim, mais est devenu indépendant de ce dernier dès la première version publique. Cependant, ces deux simulateurs ont des caractéristiques communes, en ce qui concerne leur concept d'architecture générale et leur approche de la simulation. La conception de *workload* se fait par création manuelle d'objets dédiés représentant des tâches à exécuter. Par commodité, GridSim supporte partiellement le format SWF [45] et peut lire des données d'entrée dans un format défini par l'utilisateur. Malgré ce fait, les deux simulateurs peuvent traiter une grande variété de types de *workload* : parallèles, distribués ainsi que des tâches préemptives. GridSim permet la création d'une hiérarchie de ressources simples, composées de machines physique et de processeurs. La simulation d'autres composants tels que des ressources de stockage et de réseau est possible. CloudSim met l'accent sur les ressources de calcul et fournit une couche supplémentaire de virtualisation qui agit sur l'exécution, la gestion et l'environnement des applications. Cette couche de virtualisation est responsable du processus d'approvisionnement des machines virtuelles, de la gestion du cycle de vie des machines virtuelles incluant : la création, la destruction et la migration. Il permet également l'évaluation des différentes politiques économiques en modélisant les paramètres de coûts liés à des modèles SaaS et IaaS. CloudSim fournit aussi des modèles et des entités de base pour valider et évaluer l'efficacité énergétique de différentes approches d'approvisionnement de machines virtuelles. Chaque composant participant à l'exécution des tâches peut être étendu par un objet permettant d'activité ses fonctions de gestion de consommation d'énergie. Aussi, de nouveaux modèles de puissance de machines physique, de nouveaux algorithmes de placement ainsi que de nouvelles politiques d'approvisionnement de machines virtuelles peuvent être facilement intégrés au

simulateur par l'utilisateur. Toutefois, CloudSim n'intègre pas des outils de gestion de consommation d'énergie, tel que le DVFS, permettant une gestion dynamique et précise des fréquences de CPUs.

D'autres simulateurs commerciaux permettant une étude des consommations énergétiques sont présentés dans [78, 2].

Suite à cette analyse des simulateurs de *Cloud Computing* existant, il s'avère qu'aucun ne regroupe l'ensemble des outils nécessaires pour menées des simulations énergétiques précises et utilisant la gestion dynamique de fréquence des CPUs. Un certain nombre intègrent la virtualisation et des techniques intéressantes de réduction de consommation d'énergie, mais aucun d'entre eux implémente le DVFS. C'est donc après analyse de toutes ces caractéristiques que CloudSim a été choisi pour mener les phases d'évaluation par simulation. Nonobstant le fait que le DVFS ne soit pas implémenté dans CloudSim, il regroupe déjà de nombreuses caractéristiques intéressantes.

Une présentation de l'architecture de ce simulateur, plus détaillée que celle proposée ci-dessus, est établie en Chapitre **??**. Aussi, une extension de ce simulateur est détaillée dans ce chapitre et fait l'objet d'une des contributions de cette thèse.

2.6 Bilan

Ce chapitre d'état de l'art a permis d'exposer une vue d'ensemble des différents domaines abordés dans ce manuscrit, mais aussi d'étudier de manière plus approfondie des aspects jusqu'ici un peu flous. Par exemple, la section sur la présentation générale des termes des paramètres contenus dans les SLA des fournisseurs actuels, apporte une vision globale du contenu des SLA qui démontre que la majorité des fournisseurs prêtent attention à la sécurité, la performance ou la disponibilité des services qu'ils proposent. Même si ces derniers ne donnent pas forcément des valeurs précises concernant ces paramètres, cela permet d'avoir un aperçu des paramètres intéressants à prendre en compte dans des études de QoS dans le domaine du *Cloud Computing* en se rapprochant au maximum des SLA actuels. Puis, l'analyse des travaux dédiés aux *Cloud Computing* résume les différentes caractéristiques modélisées et met également en avant les paramètres de QoS évalués dans les études actuelles. Ensuite, la présentation de différents types d'algorithmes, de méta-heuristiques et de programmation linéaire a permis d'analyser les propriétés de chacun, et ainsi, guider les choix des approches utilisées pour les travaux de cette thèse. Enfin, comme présentées dans la section précédente l'analyse et la comparaison des principaux simulateurs de *Cloud* ont mis en avant les avantages de CloudSim qui a été choisis comme outils d'évaluation. Enfin, la présentation des outils de gestion de consommation énergétique, conjointement au choix du simulateur CloudSim, a amené à se rendre compte que le DVFS se devait d'être implémenté dans ce simulateur.

Ainsi, tout cela pose les bases des travaux présentés dans la chapitres suivants.

Le chapitre suivant présente deux étapes de modélisation : tout d'abord cela concerne les modèles d'architecture matérielle et logicielle, inspirés des travaux analysés dans ce chapitre d'état de l'art, puis des modèles de paramètres Qualité de Service pouvant s'apparenter aux paramètres de QoS utilisés dans les contrats SLA des fournisseurs de services actuels également présentés précédemment dans ce chapitre.

3

Modélisation : de l'Architecture à la Qualité de Service

Sommaire

Ce troisième chapitre, intitulé "Modélisations : de l'Architecture à la Qualité de Service" va permettre de caractériser deux domaines d'environnement au sein desquels, les études présentées lors des chapitres suivant sont menées. En effet, la ou les étapes de modélisation sont indispensables afin d'introduire les caractéristiques des domaines de travail. Ainsi, ce chapitre propose une présentation du modèle d'architecture matérielle et logicielle de *Cloud Computing* pris en compte dans cette thèse, afin de préciser chaque caractéristiques des composants de ce modèle d'architecture. Aussi, une liste de modélisation de paramètres de Qualité de Service (QoS) est proposée, afin à la fois d'introduire les grandes catégories de QoS d'un environnement de *Cloud Computing*, mais aussi dans le but de définir précisément l'ensemble de ces paramètres et d'y associer, dans la mesure du possible, des métriques mesurables qui seront également utilisées dans la suite de ce manuscrit.

L'organisation de ce chapitre se fait comme suit : tout d'abord la Section 3.1 explique comment la notation des variables de modélisation ont été adoptées. La Section 3.2 présente la modélisation de l'environnement *Cloud* au sein duquel tous les travaux présentés ont été effectués. Enfin, la Section 3.3 contient une modélisation de paramètres de QoS, qui introduit des métriques mesurables et réutilisables.

3.1 Notations

Dans les définitions des modèles qui suivent, les éléments composant le système dans son ensemble sont :

- Cœur de CPU, notée c
- Machine physique, notée h
- Machine virtuelle, notée v
- Service élémentaire, noté s_e
- Service composé, noté s_c
- *Cluster*, noté k
- *Data-Center*, noté d

Certains de ces éléments peuvent occuper une certaine capacité des ressources des autres éléments, comme par exemple une machine virtuelle peut occuper une certaine proportion de ressources sur une machine physique. Ces notions de consommation de ressources et de types de ressources nécessitent donc quatre variables afin de pouvoir décrire exactement quel élément consomme quelle ressource de tel type sur tel autre élément du système. Afin d'utiliser une notation consistante et permettant de décrire toutes les situations possibles, la notation suivante a été adoptée pour l'ensemble de ce chapitre :

$$X_{Y,N}^{R_1,R_2}$$

tel que,

- X désigne l'ensemble des différentes variables utilisées par la suite
- Y désigne l'ensemble des types des ressources
- N désigne l'ensemble des noms des ressources
- R_1 désigne l'ensemble des éléments consommateurs de ressources
- R_2 désigne l'ensemble des éléments sur lesquels une portion de ressource est consommée.

L'ensemble X contient donc toutes les variables utilisées pour définir les modèles qui suivent. Parmi certains de ces modèles, certains vont avoir des variables en commun notamment ces trois variables suivantes, qui décrivent leur :

- Fraction de ressource maximum demandée ou capacité maximum du composant, notée ξ
- Fraction de ressource qui leur est allouée, notée α
- Fraction de ressource utilisée, notée ω

L'ensemble $Y = \{type_1, type_2, ..., type_n\}$ regroupe tous les types de ressources. Chaque élément de l'ensemble N des ressources, pourra être d'un type $type_i \in Y$. Pour ces deux ensembles Y et N, la notation $*$ désignera n'importe quel élément de ces ensembles. Il est important de noter que si la notation $*$ est utilisée pour l'ensemble Y, alors la notation $*$ devra également être utilisée pour l'ensemble N. En effet, si l'on veut désigner n'importe quel type de ressource, alors il n'est pas logique de vouloir exprimer une ressource $N_i \in N$ spécifique dans la même notation.

Deux types de ressources [131], fluides et rigides, sont utilisés dans ce chapitre, en voici leur définition :

Définition 3.1 : Ressources fluides

Leur modification ne peut pas entraîner le système dans un état critique ou dans l'obligation de s'arrêter totalement.

Exemple 3.1 : Exemple de ressource fluide

La capacité CPU de la machine physique peut varier au cours du temps, cela peut entraîner du retard sur l'exécution des instructions, mais n'entraînera jamais une interruption totale des processus s'exécutant sur la machine physique.

Définition 3.2 : Ressources rigides

Contrairement aux ressources fluides, la variation de capacité d'une ressource rigide peut amener le système à se trouver dans un état critique irréversible, l'empêchant totalement de fonctionner.

Exemple 3.2 : Exemple de ressource rigide

La mémoire vive (RAM) est une ressource rigide. Si trop de processus sont alloués sur une machine physique de sorte que la capacité maximum de RAM de la machine physique ne soit pas suffisante par rapport à celle demandée par l'ensemble des processus, alors l'exécution de tous les processus de cette machine ne pourront pas continuer leur exécution, et l'arrêt total de la machine est obligatoire.

Ainsi, les quatre ensembles utilisés en exposant ou en indice de X, sont définis tel que :

$$Y = \{f, g\}$$

$$N = \{cpu, mem\}$$

$$R_1 = \{c, v, s, h, k\}$$

$$R_2 = \{h, k, d\}$$

Maintenant que les bases des notations sont posées, voici quelques exemples expliqués de notations des variables ξ, α et ω.

Exemple 3.3 : Exemple de notation

- $\xi_{f,cpu}^{v,h}$, désignant la fraction maximale de ressource fluide, ici le CPU, pouvant être consommée par la machine virtuelle v sur la machine physique h
- $\alpha_{g,mem}^{v,h}$, désignant la fraction maximale de ressource rigide, ici la mémoire, ayant été allouée à la machine virtuelle v sur la machine physique h

- $\omega_{*,*}^{v,h}$, désignant la fraction de ressource utilisée par la machine virtuelle v sur la machine physique h

3.2 Architecture matérielle et logicielle

Cette section présente les modèles d'architecture matérielle et logicielle de l'environnement *Cloud* définis pour cette thèse. Pour cela, il convient de présenter, un par un, les éléments qui composent cet environnement de travail. Pour chacun d'entre eux, leur présentation contient une description concise de leur rôle, puis le détail de leurs caractéristiques et propriétés intrinsèques. L'ensemble de cette section permet d'introduire les informations nécessaires pour chaque élément et précise ainsi le niveau de modélisation adopté pour chacun.

L'environnement est composé de cinq éléments : *data center*, *cluster*, machine physique, services et machine virtuelle. Chaque composant est décrit séparément dans un tableau répertoriant toutes ces caractéristiques et propriétés.

3.2.1 Modèle de Machine Virtuelle

La machine virtuelle est le composant logiciel de base, représentant la virtualisation des ressources, et intégrant une certaine souplesse dans son utilisation. La description de sa modélisation est donc indispensable pour introduire ce qu'elle représente dans l'environnement et quels paramètres peuvent avoir une influence sur le fonctionnement du système. En effet, une machine virtuelle est responsable de l'exécution d'un service (élémentaire), présenté dans la section suivante. Les caractéristiques d'une machine virtuelle sont définis dans le Tableau 3.1 qui suit.

En plus des trois variables ξ, α et ω, une machine virtuelle est caractérisée par trois variables de temps :

- Temps de reconfiguration
- Temps de démarrage
- Temps de migration

Variable	Paramètre d'entrée	Type	Description
$\xi^{v,h}_{f,cpu}$	oui	Rationnel	Fraction de capacité CPU maximum que la machine virtuelle v peut demander sur la machine physique h
$\xi^{v,h}_{g,mem}$	oui	Rationnel	Fraction de capacité Mémoire maximum que la machine virtuelle v demande sur la machine physique h
$\alpha^{v,h}_{f,cpu}(t)$	non	Rationnel	Fraction CPU allouée à la machine virtuelle v sur la machine physique h à l'instant t
$\alpha^{v,h}_{g,mem}(t)$	non	Rationnel	Fraction Mémoire allouée à la machine virtuelle v sur la machine physique h à l'instant t
$\omega^{v,h}_{f,cpu}(t)$	non	Rationnel	Fraction CPU utilisée sur la machine virtuelle v sur la machine physique h à l'instant t
$\omega^{v,h}_{g,mem}(t)$	non	Rationnel	Fraction Mémoire utilisée sur la machine virtuelle v sur la machine physique h
$Trcf^{v,h}_{*,*}$	oui	Rationnel	Temps nécessaire à la machine virtuelle v, s'exécutant sur la machine physique h, pour changer de configuration (augmentation ou diminution de la capacité CPU ou Mémoire).
$Ton^{v,h}$	oui	Rationnel	Temps nécessaire au démarrage de la machine virtuelle v sur la machine physique h.
$Tmig^{v,(h_j,h_k)}$	oui	Rationnel	Temps nécessaire à la machine virtuelle v pour migrer depuis la machine physique h_j vers la machine physique h_k

Table 3.1 – Notations du modèle de machine virtuelle

Les notations $\xi^{v,h}_{f|g,*}$, $\alpha^{v,h}_{f|g,*}$ et $\omega^{v,h}_{f|g,*}$ pourront être utilisées par la suite pour désigner de façon plus générale la capacité maximum demandée, la fraction allouée et la fraction utilisée par la machine virtuelle v sur la machine physique h en terme de ressources *fluides* f ou de ressources *rigides* g.

3.2.2 Modèle de Services

La modélisation des services introduit deux concepts de services différents : élémentaire et composé présentés ci-dessous.

Service élémentaire

Un service élémentaire, noté s_e, fonctionne dans une seule machine virtuelle, et remplit une fonction unique bien précise :

- Service d'équilibrage de charge
- Service de calcul
- Serveur Web
- Service de base de donnée
- Service de stockage
- ...

Sa modélisation permet donc de définir ses caractéristiques, présentées en Tableau 3.2, et l'influence de celles-ci sur la ma chine virtuelle responsable de son exécution.

Variable	Paramètre d'entrée	Type	Description
v^{s_e}	non	$\oslash$	Machine virtuelle dédiée au service élémentaire s_e
$\xi^{s_e,h}_{f,cpu}$	non	Rationnel	Fraction de capacité CPU maximum que le service s_e peut demander sur la machine physique h
$\xi^{s_e,h}_{g,mem}$	non	Rationnel	Fraction de capacité Mémoire maximum que le service s_e demande sur la machine physique h
$\tau^{s_e}(t)$	non	Entier	Taux d'utilisation du service s à l'instant t, exprimé en nombre de requêtes par seconde

Table 3.2 – Notations du modèle de service élémentaire

Service composé

Un service composé représente un ensemble de services élémentaires. L'exécution de ces services élémentaires permet donc de remplir une fonction plus complexe. La modélisation d'un service composé permet de présenter ses caractéristiques (Tableau 3.3), mais aussi de définir sa topologie. La topologie d'un service composé est représentée par un graphe de tâches (DAG *Direct Acyclic Graph* en anglais), dont chaque nœud est un service élémentaire. Ce DAG permet définir des relations de dépendance entre les services élémentaires (nœuds du graphe), induisant des contraintes de communication et d'ordre d'exécution.

Variable	Paramètre d'entrée	Type	Description
S^{s_c}	oui	$\oslash$	Ensemble des services élémentaires utilisés par la service composé s_c
V^{s_c}	oui	$\oslash$	Ensemble des machines virtuelles, utilisées par le service composé s_c
nSe^{s_c}	oui	Entier	Nombre de services élémentaires utilisés par la service composé s_c
$\tau^{s}(t)$	non	Entier	Taux d'utilisation du service s à l'instant t, exprimé en nombre de requêtes par seconde
$G(V,E)^{s_c}$	oui	$\oslash$	Graphe de tâche représentant la topologie du service composé s_c

Table 3.3 – Notations du modèle de service composé

3.2.3 Modèle de Machine Physique

Une machine physique est le composant principal d'un *data center*, responsable de l'exécution des machines virtuelles. L'importance de sa modélisation réside dans le fait qu'elle permet de définir ses propriétés (Tableau 3.4) notamment en termes de ressources, de fréquences et de puissance.

Variable	Paramètre d'entrée	Type	Description
F^h	oui	$\oslash$	Ensemble des fréquences disponibles sur les cœurs des CPU de la machine physique h
nF^h	oui	Entier	Nombre de fréquences disponibles sur les cœurs des CPU de la machine physique h
nC^h	oui	Entier	Nombre de cœurs du CPU de la machine physique
$\xi^{c,h}_{f,cpu}(F_i)$	oui	Entier	Capacité maximum du cœur c du CPU de la machine physique h à la fréquence F_i
$\xi^{h,k}_{f,cpu}(t)$	non	Entier	Capacité CPU maximum de la machine physique h à l'instant t
$\xi^{h,k}_{g,mem}$	oui	Entier	Capacité Mémoire maximum de la machine physique h
$\omega^{h,k}_{f,cpu}(t)$	non	Rationnel	Fraction CPU utilisée sur la machine physique h à l'instant t
$\omega^{h,k}_{g,mem}(t)$	non	Rationnel	Fraction Mémoire utilisée sur la machine physique h à l'instant t
V^h	non	$\oslash$	Ensemble des machines virtuelles s'exécutant sur la machine physique h
n^V	non	Entier	Nombre de machines virtuelles composant l'ensemble V^h
$Pmin^{c,h}(F_i)$	oui	Entier	Puissance délivrée par le cœur c du CPU de la machine physique h à 0% d'utilisation à la fréquence F_i
$Pmax^{c,h}(F_i)$	oui	Entier	Puissance délivrée par le cœur c du CPU de la machine physique h à 100% d'utilisation à la fréquence F_i
$Pidle^{h,k}(t)$	oui	Entier	Puissance délivrée par la machine physique h à 0% de CPU à l'instant t
$Pfull^{h,k}(t)$	oui	Entier	Puissance délivrée par la machine physique h à 100% de CPU à l'instant t
$Ton^{h,k}$	oui	Rationnel	Temps nécessaire au démarrage de la machine physique h du *cluster* k.
$Toff^{h,k}$	oui	Rationnel	Temps nécessaire à l'extinction de la machine physique h du *cluster* k.

Table 3.4 – Notations du modèle de machine physique

3.2.4 Modèle de Cluster

La modélisation du composant *cluster* permet en autres de définir ses caractéristiques physiques. C'est-à-dire, s'il contient des éléments de refroidissement, différentes sortes de capteurs et l'ensemble des machines physiques qui le compose.

Dans ce modèle, un *cluster* k est défini par l'ensemble de machines physiques qu'il contient. Ce modèle de *cluster*, présenté en Tableau 3.5 considère que les machines physiques composant le *cluster* sont homogènes en terme de capacité CPU et Mémoire, peuvent fonctionner aux mêmes fréquences CPU mais délivrer des puissances différentes.

Variable	Paramètre d'entrée	Type	Description
H^k	oui	$\oslash$	Ensemble des machines physiques composant le *cluster* k
n_H	oui	Entier	Nombre de machines physiques composant l'ensemble H^k
$Tmig^{v,k}$	non	Entier	Temps de migration d'une machine virtuelle entre deux machines physiques du *cluster* k
$Tpow^{h,k}$	non	Entier	Temps nécessaire à l'allumage ou à l'extinction d'une machine physique du *cluster* k
$Ptot^k(t)$	non	Entier	Puissance totale délivrée par le *cluster* k à l'instant t

Table 3.5 – Notation du modèle de *cluster*

3.2.5 Modèle de Data Center

Un *data center*, présenté en Tableau 3.6 est l'unité la plus globale et englobe tous les modèles définis précédemment. Il est nécessaire de préciser ses caractéristiques de modélisation, car cela définit d'une certaine manière l'ensemble des composants de l'environnement considéré.

Un *data center* est composé de *clusters* hétérogènes, c'est-à-dire que les machines physiques peuvent être différentes d'un *cluster* à un autre et ont donc des caractéristiques de performance, de fréquence et de puissance différentes.

Variable	Paramètre d'entrée	Type	Description
K^d	oui	$\oslash$	Ensemble des *clusters* composant le *Data Center* K
n_K	oui	Entier	Nombre de *clusters* composant l'ensemble H^K
$P^d(t)$	non	Entier	Puissance totale délivrée par le *cluster* K à l'instant t
$Tpow^k$	non	Entier	Temps nécessaire à l'allumage ou à l'extinction d'un *cluster*

Table 3.6 – Notation du modèle de *data center*

3.3 Qualité de Service

Les fournisseurs de services *Cloud* [106, 123] peuvent proposer beaucoup de solutions différentes aux entreprises voulant se tourner vers le *Cloud Computing* [94, 8, 115]. On peut facilement comprendre que chaque solution propose différents niveaux de performance en termes de fonctionnalités, de temps de réponse ou de précision. C'est alors aux entreprises de comparer celles-ci, de comprendre comment leurs applications peuvent fonctionner sur chacune d'entre elles et enfin de trouver quelle solution est la mieux adaptée à leurs besoins.

Le contrat liant l'utilisateur d'un *Cloud* à son fournisseur de service est appelé SLA (***S**ervice **L**evel **A**greement*). Un SLA décrit le niveau de prestation, conditions d'utilisation et d'exécution, les responsabilités incombant à chaque partie (utilisateur/fournisseur), puis des informations plus précises de performance appelées SLO (***S**ervice **L**evel **O**bjective*) que le fournisseur de service se promet de respecter pour garantir une certaine qualité de service à son utilisateur.

L'expression de la Qualité de Service (QoS) [82] dans un environnement de *Cloud Computing* s'exprime avec des paramètres de plus ou moins haut niveau. Ce chapitre définit les différents paramètres de QoS au sein d'un *Cloud* en les classant dans quatre catégories :

- Performance (*Performance*)
- Sûreté de fonctionnement (*Dependability*)
- Sécurité & Données (*Security & Data*)
- Coût (*Cost*)

De nombreuses études ont déjà été menées dans le domaine de la QoS, notamment pour les services web *web services* [114, 90, 113].

Tout d'abord, deux types des paramètres sont à distinguer [112, 31] :

Définition 3.3 : Fonctionnel

Une exigence fonctionnelle permet de caractériser un comportement d'un système. Ces comportements peuvent s'exprimer sous la forme de services, de tâches ou bien de fonctions que le système est censé fournir. Il est important de pouvoir distinguer les fonctionnalités de base, qui peuvent être communes à plusieurs systèmes, et les fonctionnalités plus spécifiques de chacun qui permettent de les différencier mais aussi de les mettre en concurrence.

Définition 3.4 : Non-Fonctionnel

Une exigence non-fonctionnelle est une exigence qui spécifie des critères qui peuvent être utilisés pour juger le fonctionnement d'un système, plutôt que des comportements spécifiques. Les exigences non fonctionnelles peuvent aussi être appelées “qualités”, “contraintes”, “attributs de qualité”, “objectifs qualité”

et "exigences non-comportementales".

Dans le cadre du *Cloud Computing*, la problématique la plus importante aux yeux de l'utilisateur est de pouvoir sélectionner le service qui lui convient le mieux en fonction de ses besoins. Du côté des fournisseurs de service, il est important de pouvoir démontrer que leurs offres de services sont intéressantes et de pouvoir assurer le bon fonctionnement de leur plate-forme. Pour cela, il est nécessaire de définir des paramètres globaux de QoS afin de pouvoir comparer, analyser et classer le niveau de QoS des différents fournisseurs de services.

Dans les définitions de paramètres de QoS qui suivent, les termes anglais des paramètres et des catégories sont également cités par souci de clarté.

3.3.1 Première catégorie : Performance (*Performance*)

Définition QoS 1 : Temps d'exécution (*Execution Time*)

Le temps d'exécution [146] dépend de la complexité de la requête à exécuter (en nombre d'instructions) et de la capacité de la machine virtuelle dans laquelle elle est traîtée.

Soit $\xi^{v,h}_{*,*}$ la capacité maximum de la machine virtuelle v, $\omega^{v,h}_{*,*}$ la capacité réellement affectée à v, tel que $\omega^{v,h}_{*,*} \leq \xi^{v,h}_{*,*}$, et req une requête contenant n_i instructions.

Le temps d'exécution (Équation 3.3.1)ne dépend uniquement que du nombre d'instructions n_i à traiter de la requête req, et la capacité CPU de la machine virtuelle utilisée, $\omega^{v,h}_{*,*}$ et s'exprime donc ainsi :

$$Tex^{req,s} = \frac{n_i}{\omega^{v,h}_{f,cpu}} \tag{3.3.1}$$

Cela, en considérant que les besoins en mémoire des instructions à exécuter sont satisfaits, et donc que : $\xi^{v,h}_{g,mem} \geq \omega^{req,v}_{g,mem}$.

Définition QoS 2 : Latence & Débit (*Latency & Throughput*)

La latence (Équation 3.3.2) représente le temps nécessaire à la requête utilisateur pour arriver jusqu'au service concerné. Cela dépend uniquement de la performance et de l'état du réseau au moment où la requête utilisateur est envoyée au fournisseur de service.

Il est d'usage [103] de définir le temps T_{net}, dont un réseau a besoin pour envoyer un message avec n octets, de la manière suivante :

$$T_{net}(n) = l + \frac{n}{\rho} \tag{3.3.2}$$

avec l la Latence, et ρ le débit.

S'il était possible de transférer un message de 0 octet, le temps nécessaire pour envoyer ce message, noté $T_{net}(0)$, correspond à la Latence. En reprenant l'équation 3.3.2 avec $n = 0$, on obtient $T_{net}(n) = l$, l désignant la Latence. Le débit est caractérisé par le nombre d'octets par seconde qu'un réseau est capable d'envoyer de sa source à sa destination.

Définition QoS 3 : Temps de réponse (*Response Time*)

L'efficacité d'un service peut être mesurée en terme de temps de réponse, noté T_{rep}, ou autrement dit le temps nécessaire à la mise à disposition d'un service à un utilisateur, ou également le temps écoulé entre l'envoi d'une requête utilisateur et le moment ou celui-ci reçoit la réponse du service.

Le temps de réponse(Équation 3.3.4) dépend de plusieurs sous-facteurs comme le temps de réponse moyen, le temps de réponse maximum garanti par le fournisseur de ce service ou bien le pourcentage de chance que ce temps soit dépassé (échec du temps de réponse).

- Le temps de réponse moyen, noté T_{rep}^{avg} est donné par : $\sum_{i=1}^{n} \frac{T_{rep_i}}{n}$, avec T_{rep_i} le temps écoulé entre le départ de la requête utilisateur et l'arrivée de la réponse du service, et n le nombre total de requêtes émises.
- Le temps de réponse maximum, noté $T_{rep_i}^{max}$ est le temps de réponse maximum promis par le fournisseur de services du *Cloud*.
- Un échec du temps de réponse (Équation 3.3.3) est donné en pourcentage d'occasions que le temps effectif T_{rep_i} soit supérieur au temps de réponse maximum $T_{rep_i}^{max}$ promis par le fournisseur, et est noté σ_{rep}. Il est défini par :

$$\sigma_{rep} = \frac{n'}{n} \times 100 \tag{3.3.3}$$

avec n le nombre total d'accès au service et n' le nombre de fois où le temps de réponse est supérieur à la valeur maximum attendu ($T_{rep_i} > T_{rep_i}^{max}$).

Le temps de réponse est donc la somme de deux fois ce temps d'envoi et du temps d'exécution de la reqête req considérée :

$$T_{rep}^{req} = 2 \times T_{net}^{req}(n) + T_{ex}^{req} \tag{3.3.4}$$

3.3.2 Seconde catégorie : Sûreté de fonctionnement (*Dependability*)

Définition QoS 4 : Précision (*Accuracy*)

La précision d'un service est définie comme une fonction de deux paramètres :

- La fréquence de précision, notée ϕ_S
- La valeur de pertinence, notée τ_S

Elle mesure le degré de proximité des résultats par rapports aux valeurs attendues par l'utilisateur, et est notée $Ac = f(\phi_S, \tau_S)$. Par exemple, pour un service de stockage, c'est la variance du temps moyen de lecture et d'écriture. Pour un service de calcul, la précision est définie par le calcul de l'éloignement par rapport à la performance spécifiée dans le SLA. Dans le cas des ressources de calcul comme les machines virtuelles, l'indicateur le plus important pour la précision d'un service peut être le nombre de fois que celui-ci s'éloigne du niveau de performance de cellles-ci, tel que définit dans SLA.

Soit $n_{val'}^{u}$ le nombre de fois que le fournisseur de service ne peut garantir les valeurs promises à l'utilisateur. Alors , alors la fréquence de précision du service est définie par :

$$\phi_S = \sum_{u=1}^{n_u} \frac{n^u_{val'}}{n^u_{val}} \tag{3.3.5}$$

avec, n_u le nombre d'utilisateurs, et n^u_{val} le nombre total de valeurs attendues par l'utilisateur u.

L'autre indicateur de pertinence d'un service est la *valeur de pertinence* qui est définie comme suit :

$$\tau_S = \frac{\sum_{u=1}^{n_u} \sum_{i=1}^{n_\varsigma} \left| \frac{\varsigma_i'^u - \varsigma_i^u}{\varsigma_i^u} \right|}{n_u} \tag{3.3.6}$$

avec ς l'unité du service (calcul, réseau, stockage ...), n_ς le nombre de valeurs attendues, ς_i^u la valeur de l'unité de service attendue par l'utilisateur u et $\varsigma_i'^u$ la valeur de l'unité de service renvoyée à l'utilisateur u.

Définition QoS 5 : Fiabilité (*Reliability*)

La fiabilité [37], notée Re, représente la capacité d'un service à remplir ses fonctions requises sous certaines conditions et pendant un intervalle de temps donné. La mesure globale d'un service est lié au nombre de fois (par jour, semaine, mois ou année) que le service n'est plus capable de remplir son travail. La fiabilité reflète la capacité d'un service à fonctionner sans aucune perte pendant un temps donné sous des contraintes données. Cette caractéristique est donc basée sur la "durée de fonctionnement avant défaillance" ou "durée de fonctionnement entre pannes" du service, notée $MTTF$ [70, 107, 30, 53], ainsi que sur la base de connaissance des défaillances des utilisateurs.

le $MTTF$ peut s'exprimer en fonction de la "densité de panne", notée elle $f_{dp}(t)$:

$$MTTF = \int_0^\infty t f_{dp}(t) dt \tag{3.3.7}$$

avec,

$$\int_0^\infty f_{dp}(t) dt = 1 \tag{3.3.8}$$

Alors, λ représentant le "taux de défaillance" est égal à :

$$\lambda = \frac{1}{MTTF} \tag{3.3.9}$$

Si on considère que les pannes (défaillances du service) ne sont pas prédictibles et surviennent de façon totalement aléatoire, alors la fiabilité du service en fonction du temps peut s'exprimer ainsi :

$$Re(t) = e^{-\lambda t} \tag{3.3.10}$$

Définition QoS 6 : Disponibilité (*Availibility*)

La disponibilité, notée Av, est le pourcentage de temps pendant lequel un utilisateur peut accéder à un service. Cette disponibilité est la probabilité que le système soit en état de fonctionnement et est donc liée à son "taux de panne" ou "densité de panne". La fonction de disponibilité au cours du temps peut s'exprimer de la façon suivante :

$$Av = \int_0^{\infty} \frac{1}{f_{dp}(t)} dt \tag{3.3.11}$$

Une étude des défaillances dans les systèmes de calcul haute performance a été réalisée dans [125].

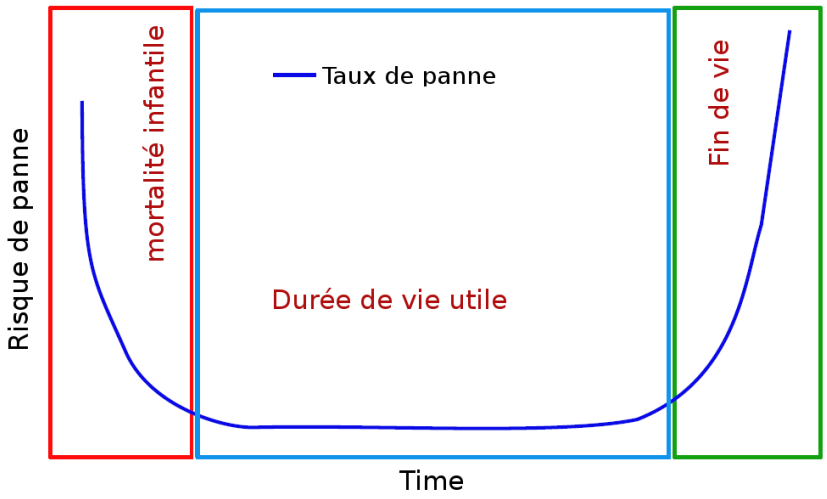


Figure 3.1 – Courbe caractéristique des risques de panne durant le cycle de vie d'un composant matériel.

Définition QoS 7 : Capacité (*Capacity*)

La capacité ξ est définie par la limite haute du nombre de requêtes (par seconde) qu'un service est capable de traiter. En terme de calcul, cela dépend de la performance des machines virtuelles utilisées par ce service et des machines physiques (composant les *cluters*) sur lesquelles tournent ces machines virtuelles.

Soit un *cluster* k composé d'un ensemble H de n_h machines (homogènes ou non), tel que $H = \{h_1, ..., h_n\}$. De plus, soit $\xi^{h_i}_{*,*}$ la capacité de chaque machine h_i (par exemple en MIPS).

Alors la capacité total, notée $\xi^{k}_{*,*}$ est égale à :

$$\xi^{k}_{*,*} = \sum_{i=0}^{n_h} \xi^{h_i}_{*,*} \tag{3.3.12}$$

avec $\xi^{h}_{*,*}$ la capacité de la machine physique h du *cluster* k,

$$\xi^{h}_{*,*} = \sum_{i=0}^{n_c} \xi^{c_i}_{*,*} \tag{3.3.13}$$

avec $\xi^{c}_{*,*}$ la capacité d'un cœur de CPU de la machine h.

Soit un service composé s_c utilisant V machines virtuelles, tel que $V = \{v_1, v_2, ..., v_n\}$ La capacité d'une machine virtuelle est désignée par $\alpha^{v,h}$, alors la capacité d'un service, $\xi^{s_c}_{*,*}s$ est égale à :

$$\xi^{s_c}_{*,*} = \sum_{i=0}^{n_v} \alpha^{v_i,h} \qquad v_i \in V \tag{3.3.14}$$

Définition QoS 8 : Extensibilité (*Scalability*)

Au sein d'une plate-forme *Cloud*, il est difficile d'estimer le nombre d'utilisateurs à un instant donné car il n'est ni prédéfini, ni constant. Sans connaître ces informations, il est donc impossible de prévoir les ressources nécessaires au fonctionnement d'un service. L'arrivée d'utilisateurs et leur intensité d'utilisation peuvent varier très rapidement et brusquement d'un instant à l'autre. Il peut donc être très difficile de satisfaire tous ces utilisateurs en même temps.

L'extensibilité [23] (aussi appelé "passage à l'échelle") représente la capacité d'augmenter la capacité de calcul et donc la capacité du système à traiter un grand nombre de requêtes utilisateurs dans un intervalle de temps donné. Elle est directement liée à la performance et s'évalue en faisant augmenter le nombre d'utilisateurs (i.e le nombre de requêtes aux services par secondes) tout en vérifiant si les caractéristiques de performance décrites dans le contrat SLA sont respectées.

La plupart du temps, l'extensibilité, notée S, s'évalue en calculant le temps de réponse d'un service (T^S_{rep}) en fonction du nombre de requêtes n_{req} que reçoit le service, comme illustré sur la figure 3.2, tel que :

$$S = f(\frac{T^S_{rep}}{n_{req}}) \tag{3.3.15}$$

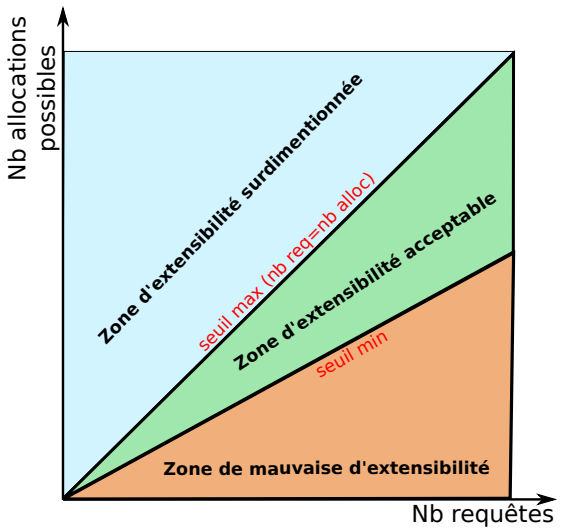


Figure 3.2 – Illustration de la relation entre le nombre de requêtes utilisateur et le nombre d'allocations possible par le fournisseur de services

L'extensibilité est donc une propriété plus que souhaitable pour un fournisseur de services *Clouds*. Cela permet de garantir la stabilité des services en cas de forte charge (croissance du nombre de services

demandés ou du nombre de requêtes des utilisateurs) de façon fluide et transparente pour les utilisateurs. Cela dit, il est dans l'intérêt du fournisseur de service de ne pas sur-dimensionner la capacité de calcul de ses machines et également de ne pas les utiliser à 100% de leur capacité pour des raisons évidentes de consommation d'énergie, et ainsi rester dans une zone d'extensibilité acceptable, comme illustré sur la figure 3.2.

Définition QoS 9 : Réactivité (*Reactivity*)

La réactivité ou (élasticité), notée Ra, est la capacité d'un service à "passer à l'échelle" durant un pic de fréquentation. Cela est défini par deux variables : le temps nécessaire, $T^{s_c}_{reconf}$, pour augmenter ou réduire le service et la variation de capacité maximum du service, notée $\delta\xi^{s_c}_{*,*}$ et définie comme : $\sum_{s_e \in S^{s_c}} \delta\xi^{s_e}_{*,*}$. La capacité de traitement maximum du service est assimilée au nombre d'unités de calcul et à leurs capacités maximum respectives pouvant leur être allouées durant le pique de fréquentation.

En d'autres termes, c'est la rapidité d'un système à augmenter son degré de performance tout en restant dans sa zone d'extensibilité acceptable. Si un système réagit trop lentement à un pic de d'utilisation alors les besoins des utilisateurs ne seront plus respectés et le système sera pris en défaut. Un système qui réagit vite arrivera à traiter les besoins des utilisateurs en temps voulu, mais il est également important de ne pas surestimer la reconfiguration à faire afin de ne pas se retrouver en zone de "d'extensibilité surdimensionnée".

Soit $Trcf^{v,h}_{*,*}$ le temps nécessaire à l'augmentation ou à la diminution de la capacité d'une machine virtuelle (par exemple en terme de capacité de calcul en MIPS) et $Trcf^{s}_{*,*}$ le temps nécessaire à l'augmentation ou à la diminution de la capacité de l'ensemble des machines virtuelles utilisées par le service s.

Suivant la complexité de la reconfiguration à effectuer (reconfigurations de machines virtuelles, allumage de machines virtuelles, ou allumage de machines physiques et migration de machines virtuelles), $Trcf^{s}_{*,*}$ peut être défini différemment.

Par exemple, en prenant comme paramètre uniquement le temps de reconfiguration des machines virtuelles, alors $Trcf^{s}_{*,*}$ sera défini tel que :

$$Trcf^{s}_{*,*} = \max_{v \in V^s} \left(Trcf^{v,h}_{*,*} \right) \tag{3.3.16}$$

La variation de capacité de la machine virtuelle v entre deux instants t_1 et t_2, tel que $Trcf^{v,h}_{*,*} = t_2 - t_1$, est définie par $\Delta\xi^{v}_{*,*}$.

$$\Delta\xi^{v}_{*,*} = \xi^{v}_{*,*}(t_2) - \xi^{v}_{*,*}(t_1) \tag{3.3.17}$$

La variation totale de capacité d'un service s est définie comme la somme de toutes les variations de capacité de ses machines virtuelles :

$$\Delta\xi^{s}_{*,*} = \sum_{v=1}^{|V^s|} \Delta\xi^{v}_{*,*} \tag{3.3.18}$$

Alors, la réactivité du service s est estimée par le ratio du ratio de variation de capacité du service et du temps nécessaire à cette reconfiguration.

$$Ra = \frac{\Delta\xi^{s}_{*,*} \times \frac{1}{\Delta\xi^{s}_{*,*}(t_1)}}{Trcf^{s}_{*,*}} \tag{3.3.19}$$

Définition QoS 10 : Dynamisme (*Dynamism*)

Le Dynamisme ou "Capacité latente", noté Dyn, représente la capacité de calcul disponible sans allumer de nouvelles machines physiques. Cela amène le fournisseur de *Cloud* à ne pas consolider de manière abusive les machines virtuelles sur les machines physiques, et ainsi à garder une certaine capacité disponible en cas de pic de charge. Le dynamisme peut être évalué comme la moyenne de la capacité de CPU libre de tous les machines physiques en cours d'utilisation. Si les machines physiques ont le DVFS activé, alors le dynamisme est calculé en fonction de la capacité maximale du processeur et de la capacité actuelle utilisée :

$$Dyn = \frac{\sum_{i=0}^{n_H} \xi^{h_i,k}_{f,cpu} - \omega^{h_i,k}_{f,cpu}}{n_H} \tag{3.3.20}$$

avec f la fréquence CPU maximum autorisée et n_H le nombre de machines physiques en cours d'utilisation.

Définition QoS 11 : Robustesse (*Robustness*)

La robustesse, notée *Rob*, peut être interprétée comme la probabilité d'un service d'être affecté par une défaillance d'un composant du *Cloud*. Du point de vue du fournisseur de services, cela revient à comptabiliser en moyenne le nombre de services pouvant être potentiellement touchés par une défaillance d'une machine physique. Plus le nombre de services mis en défaut est faible, plus le fournisseur de *Cloud* peut assurer à ses utilisateurs un haut niveau de robustesse. La robustesse peut simplement être représentée comme le nombre moyen de services en cours d'utilisation sur les machines physiques :

$$Rob = \sum_{i=0}^{n_H} \frac{n_V^{h_i}}{n_H} \tag{3.3.21}$$

Définition QoS 12 : Stabilité (*Stability*)

La stabilité d'un service, notée St, est définie par les variations de performance de celui-ci. Par exemple, pour un service de stockage, c'est la variance du temps moyen de lecture et d'écriture. Pour un service de calcul, la stabilité est définie par le calcul de l'éloignement par rapport à la performance spécifiée dans le contrat SLA :

$$St = \frac{\sum_{u=1}^{n_u} \frac{\varsigma^{u}_{avg} - \varsigma^{u}_{sla}}{T}}{n_u} \tag{3.3.22}$$

avec ς l'unité de calcul, de réseau ou de stockage de la ressource, ς^u_{avg} la performance moyenne du service demandé par l'utilisateur u, ς^u_{sla} la valeur promise dans le contrat SLA, T le temps de service et n_u le nombre total d'utilisateurs.

Définition QoS 13 : Tolérance aux fautes (*Fault Tolerance*)

La résistance aux fautes [42] d'un service, notée FTs, représente sa capacité à rester accessible et en bon état de fonctionnement lorsque des anomalies interviennent. Ces anomalies peuvent être :

- Taux d'erreur d'une machine physique, noté τ_M
- Taux d'erreur sur le réseau, noté τ_N
- Taux d'erreur logiciel : paramètres d'entrée invalides, incomplets ou contradictoires, noté τ_L

Un service *Cloud* doit être en mesure d'assurer son bon fonctionnement face à ces différents cas, ce qui définit son niveau de tolérance aux fautes. Alors la tolérance aux fautes est une fonction de ses trois paramètre, soit $Rs = f(\tau_M, \tau_N, \tau_L)$.

Si l'on pose comme hypothèse que les taux d'erreur physique et de réseau rendent totalement inaccessible le service, alors il convient de prendre en compte uniquement le taux d'erreur logiciel (τ_L). Dans ce cas, la robustesse d'un service est une fonction $f(\tau L)$ comme illustré sur la Figure 3.3.

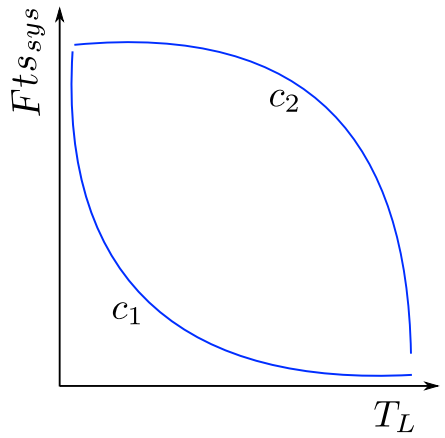


Figure 3.3 – Courbe de tolérance aux fautes d'un service *Cloud* en fonction du taux d'erreur logiciel, τ_L.

Une courbe de tolérance aux fautes se rapprochant de la courbe c_1 de la Figure 3.3 donne un service peu robuste, contrairement à un service dont la courbe serait comme c_2, qui admet un taux d'erreur élevé avant que le service soit totalement inutilisable.

Définition QoS 14 : Agilité (*Agility*)

Sous le terme agilité[1], deux différents types de caractéristiques peuvent être définis.

1. http ://www.cio.com/article/599626/Cloud_Computing_Two_Kinds_of_Agility

Le premier (1) correspond à la possibilité de variation de la disponibilité des machines physiques misent à disposition des utilisateurs du *Cloud*, le second (2) correspond à la facilité avec laquelle un fournisseur de services accepte ou non de renégocier les termes du contrat le liant à l'utilisateur.

Dans le cas (1), le principal avantage d'un environnement *Cloud* est qu'il intègre un degré d'agilité des services qui le composent. C'est à dire que l'environnement considéré peut rapidement évoluer et se développer sans engendrer des coûts insurmontables. L'agilité est mesurée comme un taux de variation de plusieurs métriques, qui démontrent la rapidité avec laquelle le système peut intégrer de nouvelles fonctionnalités. Quand on parle d'agilité de services *Cloud* on veut pouvoir comprendre à quel point le service auquel on s'intéresse est élastique, portable, adaptable et flexible. Dans ce cas, les variations possibles du système, définissant son agilité sont les suivantes :

- Nombre de ressources allouées à un utilisateur
- Flexibilité de la capacité (calcul, mémoire, ...) des machines virtuelles
- Utilisation d'un service supplémentaire

Dans le cas (2) (redéfinition des valeurs limites définies au départ dans le SLA), les paramètres à prendre en compte sont les suivants :

- Augmentation du temps de location des machines virtuelles
- Augmentation du nombre de machines mises à disposition de l'utilisateur
- Augmentation de la capacité (calcul, mémoire, ...) maximum autorisée

Définition QoS 15 : Durabilité (*Sustainability*)

La durabilité peut être définie en terme de cycle de vie du service ou bien en terme d'impact environnemental des services utilisés. On peut donc diviser ce paramètre en deux sous-attributs : durabilité de service et durabilité environnementale.

- La durabilité de service peut être définie par le nombre de composants d'un service réutilisables sans aucun changement indépendamment de l'évolution des exigences de l'utilisateur. En d'autres termes, on peut dire qu'un service est très durable s'il possède plus de caractéristiques, utilisables par l'utilisateur, que ce qui lui est nécessaire. Dans ce cas, si les exigences de l'utilisateur évoluent (augmentation du nombre de caractéristiques demandées) alors il pourra tout de même continuer à utiliser ce même service, simplement en utilisant plus de caractéristiques de celui-ci. Ceci évitant à l'utilisateur de chercher un autre service correspondant à ses critères.
  La durabilité d'un service est définie par le rapport du "*nombre de caractéristiques fournit par le service*" sur le "*nombre de caractéristiques demandés par l'utilisateur*".

  Soit $C^u_{needed} = \{c^u_1, ..., c^u_n\}$ l'ensemble des caractéristiques demandées par l'utilisateur,
  et $C^S_{provid} = \{c^S_1, ..., c^S_n\}$ l'ensemble des caractéristiques fournies par le service.
  Alors, C^u_{needed} est un sous-ensemble de C^S_{provid} , $C^u_{needed} \subset C^S_{provid}$.
  Soit N^u_{needed} et N^S_{provid} respectivement le nombre de caractéristiques du sous-ensemble C^u_{needed} et le nombre de caractéristiques de l'ensemble C^S_{provid}.

L'évolution E possible (en %) des exigences de l'utilisateur en terme de caractéristiques est donc égale à :

$$E = \frac{\left(N^{S}_{provid} - N^{u}_{needed}\right)}{N^{u}_{needed}} \times 100 \tag{3.3.23}$$

- La durabilité environnementale peut être appréciée comme le "bilan carbone" du service. Le calcul du "bilan carbone" est très complexe et dépend de beaucoup de paramètres, c'est pourquoi on préfère souvent s'attacher au calcul du PUE [105].

Définition QoS 16 : Assurance (*Insurance*)

Le terme d'assurance dans le cadre du *Cloud Computing* donne une information sur l'engagement qu'il prend à garantir le bon fonctionnement de ces services. Dans le cas contraire, des procédures de dédommagement peuvent être envisagées comme dans tout contrat d'assurance. Ce paramètre indique donc aussi la probabilité qu'un fournisseur de services *Cloud* respecte les données du contrat SLA et cela n'est pas sans effet sur le point de vue que les entreprises peuvent se faire du *Cloud Computing*. En effet, les entreprises cherchent à développer leurs activités et à fournir de meilleurs services à leurs clients. Les paramètres de fiabilité, de résistance et de stabilité sont considérés avec une grande importance par les entreprises avant de basculer de leurs applications traditionnelles vers des services de type *Cloud*.

Définition QoS 17 : Convivialité (*Usability*)

Pour une utilisation rapide de services *Cloud*, la convivialité [59] joue un rôle très important. Plus les services *Cloud* seront faciles à prendre en main plus une entreprise aura tendance à passer le cap du *Cloud Computing*. La convivialité d'un service *Cloud* intègre plusieurs paramètres comme la facilité d'accès, d'installation et d'utilisation.

Définition QoS 18 : Personnalisation (*Customization*)

La personnalisation [2] représente le pouvoir des services de *Cloud* à s'adapter aux différents types utilisateurs. En d'autres termes, chaque utilisateur doit avoir la possibilité de choisir et de sauvegarder ses propres réglages (apparence, logo, police d'écriture,...) et fichiers de configuration. Ainsi le service utilisé s'adapte à chaque utilisateur.

Définition QoS 19 : Mise à jour automatique (*Automatic Update*)

Un atout majeur d'un fournisseur de service SaaS est de pouvoir mettre à jour ses services régulièrement. La plate-forme doit donc fonctionner avec les dernières innovations en terme de service afin d'en faire directement profiter ses utilisateurs. Du point de vue utilisateur, un des atouts majeurs des mises à jour

2. http ://www.gogrid.com/news/2013/07/30/cloud-computing-cloud-customization-critical-success

automatiques, sans compter qu'elles se font de manière totalement transparente à ses yeux, est que s'il travaille en collaboration avec un autre personne utilisant cette même application, alors il n'a plus à se soucier des problèmes de compatibilité qui sont les premières sources d'erreurs dans de telles situations.

3.3.3 Troisième catégorie : Sécurité & Données (*Security & Data*)

Bien que la performance des services *Cloud* soit un paramètre très important, la sécurité [91, 134, 55, 68, 93] mise en place pour l'accès, l'utilisation des services ainsi que le stockage des données [43] est également une des préoccupations majeures des utilisateurs.

La protection des données est un aspect incontournable pour toutes les entreprises. Faire héberger ses données "à l'extérieur" est un point critique ce qui demande aux fournisseurs de services *Cloud* de mettre en place des politiques de sécurité très performantes. La confidentialité [110] des données est tout autant indispensable : l'utilisation de techniques de preuve de stockage [11, 69] permet de vérifier que le *Cloud* ne subit pas d'attaque de type "déni de service" et que toutes les données des clients sont intactes. Les données dynamiques peuvent également être vérifiées avec une approche similaire [44].

Ces techniques fonctionnent avec un très faible temps de calcul tout en transférant peu de données entre les clients et le serveur.

Définition QoS 20 : Authentification (*Authentication*)

Les moyens traditionnels d'authentification ne sont plus d'actualité dans une infrastructure de *Cloud* et c'est pourquoi de plus en plus de fournisseurs *Cloud* sont à la recherche d'un service d'authentification performant [120], appelé Authentification-As-A-Service (AAAS). Cela afin d'accroître la sécurité et gérer l'authentification des utilisateurs plus facilement.

L'authentification forte peut aujourd'hui être assurée partout où un mot de passe est utilisé par l'utilisation des normes industrielle tels que RADIUS (**R**emote **A**uthentication **D**ial-**I**n **U**ser **S**ervice) et SAML [101](**S**ecurity **A**ssertion **M**arkup **L**anguage) et la disponibilité d'API pour d'autres applications.

L'utilisation de jeton d'authentification, méthode appelée "Cloud Token" [120] permet aux utilisateurs de ne pas perdre leur jeton lorsqu'ils décident de migrer de plate-forme en plate-forme. De plus une automatisation de la gestion d'authentification permet de réduire considérablement le coût de gestion et d'administration.

Définition QoS 21 : Autorisation (*Authorization*)

Les moyennes et grandes entreprises ont généralement des besoins spécifiques de caractéristiques d'autorisation d'accès aux données pour leurs utilisateurs de *Clouds* [93]. C'est à dire l'attribution de privilèges, ou droits, aux utilisateurs en fonction de leurs rôles et de leurs besoins.

Dans certains cas, une application d'entreprise peut exiger un "contrôle d'accès à base de rôles" (RBAC) [121], dans laquelle les autorisations sont structurées de manière à répondre aux exigences des rôles fonctionnels de l'entreprise.

À ce jour, la mise en place des autorisations au sein d'un service *Cloud* et la gestion des capacités sont faibles, et ne peuvent pas être gérées avec une granularité satisfaisante. La plupart des fournisseurs de services *Clouds* prennent au moins en compte deux rôles (privilèges) : *administrateur* et *utilisateur*. Cela permet d'attribuer des droits d'administration sur le service afin de pouvoir gérer les profils utilisateurs, les politiques d'accès au service ou d'autoriser ou non les connexions au service suivant leur provenance.

Définition QoS 22 : Intégrité (*Integrity*)

La capacité de conserver l'intégrité des données[3] des utilisateurs est d'assurer que le contenu de leurs données ne soit pas modifié à leur insu. Une manière de vérifier l'intégrité d'un contenu est de comparer l'état actuel des données par rapport à un état connu dans le passé (à $t = t_b$) sans qu'il y ait eu d'intervention de l'utilisateur entre les deux moments.

Soit l'ensemble de données D_u^t, composé de n_d éléments tel que $D_u^t = \{data_0, ..., data_n\}$, représentant toutes les données appartenant à à l'instant t. L'intégrité des données est assurée par $D_u^{t_b} \equiv D_u^t$, tel que chaque élément $data_i \in D_u^t$ soit égal à l'élément $data_i \in D_u^{t_b}$.

Définition QoS 23 : Confidentialité (*Confidentiality*)

Plusieurs types de problèmes tels que les bogues logiciels, erreurs venant de l'opérateur et attaques externes peuvent compromettre la confidentialité [110] des données applicatives des *Clouds*, et de ce fait les rendre vulnérables face aux actions malveillantes de certains utilisateurs dont l'accès au *Cloud* aurait dû être interdit. Des techniques de protection [88] contre les attaques visant à détruire la confidentialité des données peuvent également aider les utilisateurs à vérifier la cohérence et la mise à jour de leur données. Ces techniques nécessitent que peu d'échange d'informations, telle que la racine d'un arbre de hachage de Merkle [95].

Définition QoS 24 : Responsabilisation (*Accountability*)

Le terme anglais *Accountability* [77] peut être associé au terme *monitoring* et être traduit par "Responsabilisation", "Vérification" ou "Contrôle" en français.

Au sein d'un *Cloud* lorsque l'on parle d'*accountability*, il est d'usage de parler de "chaîne de vérification" ou de "chaîne de contrôle", de différents paramètres, que le fournisseur de service met en place au sein de son *Cloud*.

Les fournisseurs de services implémentent donc des mécanismes de vérification, essentiellement destinés à assurer un niveau de sécurité sur les données des utilisateurs. Cela dans l'optique de pouvoir leur fournir un suivi et un contrôle de leurs données stockées dans le *Cloud*.

Un exemple de maillons de chaîne de contrôle est visible sur la Figure 3.4.

Une chaîne de contrôle d'un fournisseur de *Cloud* permet de garantir l'accessibilité aux outils suivants :

3. http ://www.jatit.org/volumes/Vol58No3/12Vol58No3.pdf

- Des outils permettant de donner aux utilisateurs un contrôle et une vision de la façon dont leurs données sont utilisées, l'assurance que leurs données soient traitées et protégées de la manière dont ils le souhaitent.
- Des outils permettant aux utilisateurs de faire des choix sur la façon dont les fournisseurs de services de *Cloud* peuvent utiliser et protéger leurs données, d'être informés sur les risques, les conséquences et la mise en place de ces choix.
- Des outils pour vérifier que les attentes en terme de sécurité des utilisateurs soient respectées, et qu'il n'y ait pas d'entrave au règles définies dans les SLA.
- Des outils d'explication et d'aide pour informer l'utilisateur sur la façon dont est effectué la surveillance et le contrôle de leurs données, par le fournisseur de service d'un point de vue éthique.

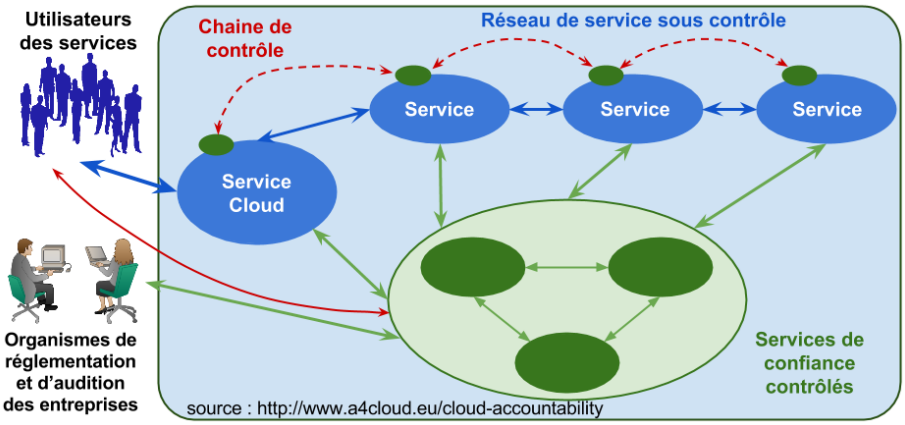


Figure 3.4 – Illustration d'une chaîne de contrôle mise en place par un fournisseur de services *Cloud*.

Définition QoS 25 : Traçabilité (*Traceability*)

D'une manière générale la traçabilité est définie comme suit [4] : *"La traçabilité désigne la situation où l'on dispose de l'information nécessaire et suffisante pour connaître (éventuellement de façon rétrospective) la composition d'un produit tout au long de sa chaîne de production et de distribution."*

Au sein d'un *Cloud*, la traçabilité de l'utilisation des services permet à l'utilisateur de savoir précisément comment le service est utilisé. Toute information relative aux interventions effectuées concernant de près ou de loin le ou les services utilisés doivent être pouvoir mise à disposition de chaque utilisateur. En effet, l'impossibilité de connaître l'état de fonctionnement, ou la localisation des données est un réel frein à l'utilisation de ces services *Clouds* [148]. Il est donc important d'informer les utilisateurs, à travers un journal de logs (traces écrites), quels types d'actions automatisées ou humaines ont été effectuées, et où sont stockées leurs données [5] [98].

Une plate-forme de traçabilité permet de répondre à ces problèmes de non-transparence, et donc de sauvegarder une chaîne d'événements indiquant par exemple les opérations humaines, les transferts de fichiers, et l'activité des processus automatiques ainsi que des informations provenant des systèmes connexes

4. http ://fr.wikipedia.org/wiki/traçabilité
5. http ://www.hpl.hp.com/techreports/2011/HPL-2011-38.pdf

tels que les systèmes d'authentification et de gestion des équipements.

Définition QoS 26 : Cryptage (*Encryption*)

Le cryptage[99] est un procédé utilisé pour protéger le transit ou le stockage d'informations. Il s'agit de la conversion des données en cryptogrammes, qui ainsi ne peuvent être lus que par les personnes propriétaires de ces données ou autorisées à en lire le contenu. Le chiffrement est utilisé pour protéger les informations sensibles stockées et utilisées dans les réseaux, appareils mobiles ou sans fil.

Dans un *Cloud*, les algorithmes de chiffrement sont utilisés pour protéger les données sortantes, afin que les informations ne soient pas vulnérables une fois qu'elles sont à l'extérieur de l'entreprise de laquelle elles appartiennent. Le chiffrement est couramment utilisé pour s'assurer de répondre aux réglementations des entreprises. HIPAA (*Health Insurance Portability and Accountability Act Compliance Report*) et PCI DSS (*Payment Card Industry - Data Security Standards*) sont des outils essentiels de sécurisation des données *Clouds* pour les entreprises utilisant des applications SaaS tels que Salesforce.com ou Oracle CRM On Demand.

Définition QoS 27 : Isolation (*Isolation*)

Dans les *Clouds*, un défi est de permettre aux utilisateurs authentifiés de consulter leurs propres données depuis l'extérieur de ce *Cloud* tout en empêchant un utilisateur malveillant d'exploiter une faille du prestataire de services lui permettant d'accéder aux données des utilisateurs et rendant par la même occasion l'ensemble du système vulnérable. Pour éviter ce problème, la mise en place d'un service de journalisation (*log* en anglais) sécurisé peut aider à isoler les données des différents utilisateurs.

Définition QoS 28 : Cycle de vie des données (*Data life cycle*)

Oracle [104] définit le cycle de vie des données par la fréquence d'accès à celles-ci. Plus le temps passe, plus la fréquence d'accès aux données diminue, menant finalement à un archivage de celles-ci. Le cycle de vie comprend trois états : actif, moins actif et état historique ou archivé. La *Cloud Security Alliance* [5] a proposé le concept de cycle de vie de sécurité des données, qui résume le cycle de vie de la sécurité des données en six étapes :

1 : Production
2 : Conservation
3 : Utilisation
4 : Partage
5 : Archivage
6 : Destruction

Définition QoS 29 : Non-Répudiation (*Non-Repudiation*)

La non-répudiation [147] implique le fait que le contrat (ici de sécurité) ne peut être remis en cause par aucune des deux parties (utilisateur/fournisseur).

3.3.4 Quatrième catégorie : Coûts (*Cost*)

Définition QoS 30 : Coût de service (*Service Cost*)

La première question qui surgit, notamment au sein d'une entreprise, avant de se tourner vers le *Cloud Computing* est : "*cela est-il rentable ou non*" ? La rentabilité et le coût sont clairement des attributs qui ont une très grande d'importance au yeux des utilisateurs (particulier ou entreprise) de *Cloud*. Le coût tend à être un paramètre facilement quantifiable, mais il est important de pouvoir l'exprimer avec des caractéristiques pertinentes par rapport à l'organisation à laquelle on s'adresse. Pour un fournisseur de service *Cloud* le coût peut s'exprimer ainsi :

γ_v^{cap} : Coût de location à l'heure d'une machine virtuelle v de capacité *cap*.

ΔT_{loc} : Durée d'utilisation en heure du service.

N_v : Nombre de machines virtuelles demandées.

Alors, le coût final que l'utilisateur aura à payer peut se définir logiquement ainsi :

$$\gamma_{tot} = \gamma_v^{cap} \times N_v \times \Delta T_{loc} \tag{3.3.24}$$

Définition QoS 31 : Coût énergétique (*Energy Cost*)

Le coût énergétique d'un équipement, exprimé en kWh, représente l'énergie nécessaire pour le faire fonctionner sur une période de temps donnée. Il est fonction de la puissance (en Watt) délivrée par cet équipement (machine physique par exemple) et de la durée d'utilisation (en Heure). Concernant la puissance délivrée par une machine physique, plusieurs paramètres rentrent en jeux, notamment le taux d'utilisation du CPU. L'utilisation du modèle affine, bien que assez simpliste, a fait ses preuves depuis longtemps et est utilisé dans de nombreux travaux de recherche. En considérant qu'un CPU de machine physique peut fonctionner à différentes fréquences, le modèle affine [116] pour une fréquence CPU F_i fixée s'exprime de la façon suivante :

$$P(F_i) = \alpha \left(P_{full}(F_i) - P_{idle}(F_i)\right) + P_{idle}(F_i) \tag{3.3.25}$$

avec, $P_{full}(F_i)$ and $P_{idle}(F_i)$ les puissances délivrées par la machine physique concernée, aux taux d'utilisation CPU respectifs de 0% et de 100% et fonctionnant à la fréquence F_i.

La précision de ce modèle, utilisé avec le DVFS a fait l'objet d'une étude présentée dans cette thèse, ainsi que dans l'article [61]. Autrement, d'autres modèles énergétiques plus pointus peuvent être utilisés, comme présenté par Da Costa *et. al.* [36].

Définition QoS 32 : Empreinte carbone (*Carbon Cost*)

L'empreinte Carbone [145] représente l'impact que peut avoir un système sur l'environnement en tenant compte du volume de dioxyde de carbone (CO2) émis par combustion d'énergies fossiles. Dans les *Clouds*, les valeurs et les coûts qui lui sont associés peuvent varier en fonction des fournisseurs de services, car de nombreux paramètres rentrent en jeux. Selon la *Open Data Center Alliance*[6], ces variations peuvent être dues à :

- L'efficacité énergétique du matériel utilisé
- La configuration et les paramètres du matériel et des logiciels utilisés
- Le degré de virtualisation et de l'efficacité de sa gestion
- Les conditions environnementales (par exemple, climat nordique vs climat méditerranéen)
- L'efficacité de l'infrastructure du *Data Center* et de son hébergement (i.e., efficacité énergétique ou PUE)
- Source d'électricité utilisée : charbon, nucléaire, générateurs locaux, etc...
- La "compensation carbone" *Carbon offsetting* [142, 122] (engagement à diminuer les émissions de carbone)
- Les dispositions nationales ou régionales régulant les taxes liées aux émissions de carbone.

Ainsi, la quantité de carbone produit peut s'exprimer comme suit :

$$CFP = IT_{eq} \times E_{eq} \times E_{ovh} \times (Ca_{em}(src) + Lo) \tag{3.3.26}$$

avec, IT_{eq} le nombre d'équipement utilisé, E_{eq} la consommation énergétique induite par ces équipements, E_{ovh} le surcout énergétique, $Ca_{em}(src)$ les sources de production d'électricité et donc d'émission de carbone dans l'atmosphère, également appelées CEF (**C**arbon **E**mission **F**actor), et Lo représente les pertes dues au transport de l'électricité de sa source de production à sa destination consommatrice.

6. http ://www.opendatacenteralliance.org/docs/Carbon_Footprint_and_Energy_Efficiency_Rev2.0.pdf

3.4 Expressivités et limites

Catégorie	Nom	Notation	Fonctionnel
PERFORMANCE	Execution Time	T_{ex}	non
	Latency	α	non
	Throughput	β	non
	Response Time	T_{rep}	non
DEPENDABILITY	Accuracy	$Ac = f(\phi_S, \tau_S)$	non
	Reliability	Re	non
	Availability	A	non
	Capacity	C_{Clust}^{Tot}	non
	Scalability	S	non
	Reactivity	Ra	non
	Dynamism	Dyn	non
	Robustness	Rob	non
	Stability	$St(t_1, t_2)$	non
	Fault Tolerance	FTs	non
	Sustainability	E	non
	Agility	$\oslash$	oui
	Insurance	$\oslash$	oui
	Usability	$\oslash$	oui
	Customization	$\oslash$	oui
	Automatic Update	$\oslash$	oui
SECURITY	Authentication	$\oslash$	oui
	Authorization	$\oslash$	oui
	Integrity	$\oslash$	oui
	Confidentiality	$\oslash$	oui
	Accountability	$\oslash$	oui
	Traceability	$\oslash$	oui
	Encryption	$\oslash$	oui
	Data Life Cycle	$\oslash$	oui
	Non-Repudiation	$\oslash$	oui
COST	Service Cost	ω_{tot}^{u}	non
	Energy Cost	$P^h(F_i)$	non
	Carbon Cost	CFP	non

Table 3.7 – Tableau récapitulatif des notations des paramètres de QoS

La confection de modèle est toujours une étape délicate. En effet, cela impose de choisir le degré de modélisation nécessaire pour les travaux engagés. Tout d'abord en terme de nombre de composants du modèle mais aussi en terme de granularité. Des modèles d'une granularité très fine peuvent permettre de modéliser beaucoup de caractéristiques et ainsi exprimer ceux-ci d'une manière très détaillée. Mais plus le niveau de granularité est élevé, plus la clarté des modèles peut être affectée et ainsi dégrader la compréhension de ceux-ci. Il convient donc d'adopter l'approche qui permettra de représenter tous les composants indispensables aux modèles, et définir une granularité des caractéristiques s'adaptant aux niveaux de détails nécessaires dans les phases expérimentales.

Les modèles d'architecture matérielle et logicielle ont été définis de manière à exprimer comment chaque composant est représenté dans la suite des travaux présentés. Ainsi, tout l'environnement ainsi que les caractéristiques de chaque composant sont définis pour l'ensemble des études qui suivent. Par exemple, dans le cas du modèle de machine physique il était indispensable d'intégrer la caractéristique d'acceptation de plusieurs fréquences de fonctionnement des CPUs afin d'être en mesure par la suite d'exprimer des équations ou des contraintes représentant l'utilisation du DVFS. Aussi, le concept de reconfiguration de machines virtuelles en terme de capacité de traitement CPU se devait d'être modélisé afin de pouvoir exprimer son utilisation dans les différentes expériences mettant en jeu cette propriété des machines virtuelles.

Les modèles de paramètres de Qualité de Service sont présentés en 4 catégories (et récapitulé en Tableau 3.7). Cela permet de mettre en avant les principales catégories de QoS au sein d'un environnement de *Cloud Computing*, mais aussi de séparer ces paramètres les uns des autres. Ainsi, au sein de chacune des catégories présentées, chaque paramètre peut être défini de manière précise. Cette section de modélisation de QoS exprime le sens de chacun des paramètres, et rend mesurable certain d'entre eux par la définition de métriques. Les limites de cette modélisation est induite par les paramètres de type fonctionnel qui ne peuvent être représentés mathématiquement.

Le chapitre suivant présente comment ces modèles ont été exploités, notamment par l'utilisation d'algorithmes et de méta-heuristiques de placement.

4

Optimisation multi-objectifs de qualité de service Cloud

Sommaire

Ce chapitre expose pourquoi et comment la modélisation de la qualité de service, proposée en Section 3.3, a été utilisée au sein d'une approche d'optimisation multi-critères. Les paramètres et les métriques définis ont pour but d'apporter une analyse plus large et à la fois plus précise des enjeux mis en évidence dans les travaux dédiés aux algorithmes de placement dans un *Cloud*. Trouver la solution optimale à un problème de placement étant irréalisable en temps polynomial autant dans un cas général [51], que dans des cas plus particuliers [87] [136], des méta-heuristiques se rapprochant plus ou moins de la solution optimale sont utilisées afin d'obtenir une solution satisfaisante. Il est également important de rappeler que, outre les paramètres de qualité de service sélectionnés, les reconfigurations de machines virtuelles ainsi que le DVFS font parties intégrantes du problème d'ordonnancement exposé dans ce manuscrit, ce qui amplifie nettement la complexité de celui-ci.

Tout d'abord, une modélisation par un algorithme génétique est proposée. Cet algorithme génétique dédié au problème de placement de machines virtuelles dans un *Cloud* a été implémenté dans deux versions différentes. La première tient uniquement compte de services élémentaires, alors que la seconde utilise des services composés. Cette version complexifie nettement la résolution du placement, notamment car un aspect temporel doit être intégré à la résolution. En effet, la topologie d'un service complexe, la plus simpliste soit-elle, induit une dépendance entre les services que l'algorithme génétique doit prendre en

compte pour obtenir une solution de placement valide. Ces deux versions de l'algorithme génétique sont présentées dans la Section 4.4. Aussi, deux algorithmes gloutons, *Round Robin* et *Best-Fit Sorted* ont été utilisés afin de permettre une comparaison avec l'algorithme génétique. Les caractéristiques de ces deux algorithmes sont présentées en Section 4.3.

Il est important de préciser ici que la version de l'algorithme utilisant des services composés est présentée comme un travail préliminaire appelant de futurs travaux. Cette version de l'algorithme génétique n'est pas utilisée dans les autres travaux exposés dans la suite de ce manuscrit mais présente un intérêt certain grâce aux complexités qu'elle soulève.

La première section de ce chapitre expose comment l'utilisation de plusieurs métriques de qualité de service au sein d'algorithmes d'ordonnancement peuvent à la fois enrichir les études de recherche actuelles, améliorer les approches multi-objectifs dédiées au *Cloud Computing*, mais aussi mettre en avant la perspicacité parfois ignorée de certains algorithmes aux comportements plus basiques.

4.1 Enrichissements apportés par une approche multi-critères de QoS Cloud

Dans la majorité des études de recherche, les algorithmes de placement sont basés sur l'optimisation de la consommation d'énergie et du temps de réponse, interprétés différemment suivant les domaines concernés. Dans le domaine du *Cloud Computing*, certaines autres métriques sont prises en compte, tel que, d'un point de vue utilisateur, le coût financier qu'engendre une location de machine virtuelle pendant plusieurs heures, ou d'un point de vue du fournisseur de services, les bénéfices qu'ils peuvent tirer de ce processus.

L'analyse des propositions de SLA des principaux fournisseurs de services SaaS actuels, faite en Section 2.4, permet de prendre conscience que l'inclusion, même d'un seul de ces paramètres qui sert directement l'intérêt du fournisseur de services, dans les travaux de recherche sur les algorithmes de placement dédiés au *Cloud Computing*, pourrait permettre des études plus approfondies. Par exemple, si le paramètre *latent capacity*, cité dans la proposition de SLA de Oracle était utilisé comme une métrique à optimiser dans les algorithmes de placement, en complément de la consommation énergétique et du temps de réponse, cela amènerait à analyser un compromis intéressant entre la réduction de consommation énergétique, la performance voulue à un instant t et la capacité des services à réagir si un pic d'utilisation (augmentation brusque du nombre de requêtes) se produisait. En effet, prendre en compte plus de paramètres de QoS, assimilés à des métriques d'optimisation dans des algorithmes de placement visant à réduire l'énergie consommée, peut mener à choisir une configuration temporaire de consolidation des machines virtuelles défavorables pour certaines métriques, mais garantissant un haut niveau de performance pour d'autres métriques qui sont pour la plupart ignorées dans les études du moment. L'utilisation de plusieurs métriques permet ainsi d'avoir une vision plus complète de l'état de l'ensemble des ressources physiques, et apporte aux fournisseurs de services une possibilité d'analyse plus élaborée ainsi qu'un nombre plus important de solutions de configurations.

De plus, une optimisation multi-critères permet également d'analyser l'influence des paramètres les uns par rapport aux autres. Cela, afin d'estimer la manière dont chaque paramètre influence l'optimisation, d'analyser l'antagonisme provoquer par l'utilisation conjointe de ces métriques mais également d'en vérifier leur pertinence. Ces différents points ne sont pas sans importance car il n'est pas forcément chose aisée de sélectionner un ensemble de paramètres de QoS ayant à la fois une réelle pertinence pour l'analyse du fonctionnement du système et dont l'optimisation conjointe avec d'autres paramètres résulte en un compromis satisfaisant.

Un autre avantage intéressant d'une analyse simultanée de paramètres de QoS *Cloud* divers peut faire ressortir des propriétés à priori non démontrées d'algorithmes gloutons dont le comportement intrinsèque n'autorise pas une optimisation directe des métriques analysées.

4.2 Sélection des métriques de qualité de service

Quatre paramètres de QoS ont été choisis parmi tous ceux présentés en Section 3.3 : la consommation d'énergie qui permet de tenir compte des problèmes environnementaux, le temps de réponse qui permet d'avoir une mesure de performance pure, la robustesse qui rassure les fournisseurs de services et les utilisateurs sur la probabilité d'être concernés par une défaillance du système, et le dynamisme qui assure une certaine réserve de performance en cas de pic de trafic.

Afin de pouvoir mesurer et évaluer ces métriques dans les deux approches proposées dans ce chapitre, quelques modifications ont dû être apportées aux définitions des métriques utilisées par rapport à celles présentées de façon générique en Section 3.3. Les métriques, telles qu'utilisées par les algorithmes de placement, sont décrites ci dessous :

- **Temps de réponse** : La métrique de Temps de Réponse est assimilée au temps d'exécution de chaque machine physique. Ainsi, le Temps de Réponse (en secondes) (Équation 4.2.1) est calculé en fonction de la capacité (en MIPS) de chaque machine virtuelle et du nombre d'instruction (en Million d'Instructions) qu'elles doivent exécuter :

$$T_{rep}^{h,k} = \max\left(\frac{NbInstr^{v}}{\omega_{f,cpu}^{v,h}}\right) \tag{4.2.1}$$

  avec $NbInstr^{v}$ le nombre d'instructions à exécuter par la machine virtuelle v. Cette métrique doit être minimisée afin de minimiser le Temps de Réponse.

- **Énergie** : La métrique de consommation énergétique est calculée en utilisant la formule de puissance (Équation 4.2.2) donnée dans la définition du paramètre *Energy Cost* :

$$P(F_i) = \alpha\left(P_{full}(F_i) - P_{idle}(F_i)\right) + P_{idle}(F_i) \tag{4.2.2}$$

  Afin d'obtenir la valeur de consommation exacte pour chaque machine physique, celle-ci est calculée en fonction des variations de puissance délivrée par chaque machine physique au cours du temps (en fonction des fins d'exécution des machines virtuelles) et du temps total d'exécution (ou temps

de réponse) noté $T_{rep}^{h,k}$. La métrique de consommation d'énergie (en Wh) (Équation 4.2.3) s'exprime ainsi :

$$E = \sum_{i=1}^{n_H} P(F_i)^{h_i} \times T_{rep}^{h,k} \tag{4.2.3}$$

Cette métrique doit être minimisée afin de minimiser la consommation énergétique.

- **Robustesse** : La métrique de Robustesse est assimilée au nombre de machines virtuelles pouvant être affectées par une défaillance (panne logicielle, panne physique, panne réseau, etc...) de machine physique. Celle-ci (Équation 4.2.4) est donc représentée par le nombre moyen de machines virtuelles allouées sur chacune des machines physiques :

$$Rob = \sum_{i=0}^{n_H} \frac{n_V^{h_i}}{n_H} \tag{4.2.4}$$

Cette métrique doit être minimisée afin de maximiser la Robustesse.

- **Dynamisme** : La métrique de Dynamisme (ou Capacité Latente) (Équation 4.2.5) représente la moyenne de capacité CPU libre sur chacune des machines physiques allumées pouvant être utilisée s'il y a une augmentation très rapide du nombre des requêtes à traiter. Elle est donc exprimée ainsi (en nombre MIPS libres par machine) :

$$Dyn = \frac{\sum_{i=0}^{n_H} \xi_{f,cpu}^{h_i,k} - \omega_{f,cpu}^{h_i,k}}{n_H} \tag{4.2.5}$$

Cette métrique doit être maximisée afin de maximiser le Dynamisme.

4.3 Algorithmes gloutons

Cette section présente les algorithmes gloutons utilisés, décrits ci-après, en complément de l'algorithme génétique présenté en détail en Section 4.4. Ces deux algorithmes gloutons sont : *Round-Robin* et *Best-Fit Sorted*. Leur fonctionnement est plus basique par rapport à un algorithme génétique par le fait qu'ils n'intègrent pas une optimisation directe des métriques de QoS. Cependant, l'analyse multi-critères de leurs résultats permet d'interpréter leur perspicacité d'une autre manière.

4.3.1 Round-Robin

L'algorithme *Round-Robin* (RR) était empiriquement utilisé pour l'ordonnancement dans les réseaux, avec aucune priorité sur la destination choisie, lorsque le concept d'environnement virtualisé n'existait pas encore. Son utilisation dans le contexte de ce chapitre peut paraître assez étrange étant donné que le principe de cet algorithme ne présente aucune intelligence particulière qui permettrait de tirer profit de la virtualisation et donc de la notion de consolidation machines virtuelles, actuellement couramment utilisée

dans les *Data Centers* de *Cloud* afin de minimiser la consommation d'énergie. Ainsi, son utilisation dans ce chapitre est analysée en tenant compte de plusieurs paramètres, et pas seulement la consommation énergétique, ce qui permet d'en dégager une analyse plus élaborée et aussi démontrer ses avantages.

Le *Round-Robin* implémenté trie les machines physiques en ordre croissant en fonction leur type. C'est à dire que l'algorithme parcourra les machines physiques de type 0 en premier, puis celle de type 1, etc... Dû au comportement intrinsèque de l'algorithme ce tri a peu d'importance lorsqu'un grand nombre de machines virtuelles doit être alloué car des machines virtuelles seront quand même placées sur des machines physiques énergétiquement mauvaises. Mais plus le nombre de machines virtuelles à allouer diminue plus ce tri devient intéressant. Cependant ce genre de tri n'est pas très efficace avec Round-Robin tant que le nombre de machines virtuelles à allouer est supérieur au nombre de machines physiques. En effet, celles-ci seront de toute façon utilisées quelque soit leur type.

4.3.2 Best-Fit Sorted

L'algorithme *Best-Fit Sorted* (BFS) est plus connu grâce à sa capacité à profiter efficacement d'un environnement virtualisé. Utilisé avec des tris effectués sur les caractéristiques des machines physiques et des machines virtuelles, le BFS obtient de bons résultats en termes de consommation d'énergie.

Le BFS implémenté exécute deux tris différents sur les machines physiques. Dans un premier temps, les machines physiques sont triées en fonction de leur type. Une fois ce premier tri terminé, un deuxième tri est effectué en fonction de la quantité (décroissante) de MIPS libres sur chacune des machines physiques. Cela permet d'allouer les machines virtuelles sur les machines physiques les plus efficaces énergétiquement mais également de tenter de consolider du mieux que possible les machines virtuelles.

4.4 Algorithme génétique

Un Algorithme Génétique [130], en anglais *Genetic Algorithm* (GA), est une méta-heuristique d'optimisation qui imite l'évolution naturelle. Un algorithme génétique utilise un ensemble d'individus, appelé population, dans lequel chaque individu (aussi dénommé chromosome) représente une solution au problème à résoudre. Le principe de base d'un algorithme génétique est de faire évoluer cette population initiale, sur un certain nombre de générations, pour aboutir à une population qui contiendra des chromosomes meilleurs que ceux de départ. D'une génération à une autre, des opérateurs spécifiques (opérateurs génétiques) sont appliqués sur chaque individu afin d'explorer de nouvelles solutions possibles. À chaque génération, suite à l'application de ces opérateurs génétiques qui génèrent de nouveaux individus, tous les chromosomes (initiaux et nouveaux) sont triés en fonction de leur valeur de *fitness*. Chaque individu se voit attribuer un score (valeur de *fitness*), suite au calcul d'une fonction objectif. Chaque individu est donc évalué en fonction de cette valeur, et seuls les chromosomes ayant obtenus une valeur de *fitness* estimée être assez bonne pour faire partie de la génération sont retenus pour constituer la population de travail de la génération suivante. Ainsi, à chaque génération le jeu des opérateurs créant de nouveaux chromosomes tend à ce que la population contienne des individus représentant de meilleures solutions. Un algorithme génétique est donc une méta-heuristique, pouvant s'adapter à plusieurs types de problème, et faisant évoluer une population de départ en appliquant aléatoirement des opérateurs rendant possible le parcours de l'espace de solutions. Il est important de rappeler que la manière dont un algorithme génétique exécute sa recherche de solution

est appliquée de manière à explorer le mieux possible l'espace de solutions, mais ne garantit en rien la qualité de la solution trouvée (par rapport à l'optimal).

Dans ce chapitre, l'algorithme génétique présenté est dédié à la résolution du problème d'allocation de machines virtuelles dans les *Clouds*. Considérant qu'une solution au problème à résoudre est le résultat d'une optimisation multi-objectifs, les valeurs de *fitness* de chacun des chromosomes permettant de juger la qualité d'une solution, sont calculées en tenant compte de chacune des métriques de QoS sélectionnées.

La modélisation, en terme de chromosomes et de gènes correspondant au problème d'ordonnancement de machines virtuelles sur un ensemble de machines physiques est décrite dans la section suivante.

4.4.1 Modélisation

Un algorithme génétique étant par définition une méta-heuristique pouvant être adaptée à de nombreux types de problèmes. Une description claire de la modélisation choisie dans le contexte de cette thèse est donc nécessaire :

- Un chromosome représente une solution de placement des machines virtuelles, comme illustré en Figure 4.1.
- Un gène représente une machine virtuelle.
- La valeur attribuée à un gène représente le numéro de la machine physique sur laquelle la machine virtuelle a été allouée.
- En parallèle, les caractéristiques des machines virtuelles et des machines physiques sont sauvegardées de manière à pouvoir calculer les valeurs de chacune des métriques ainsi que la valeur de *fitness* des chromosomes.

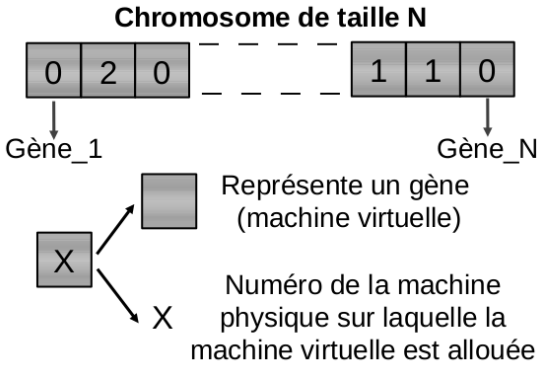


Figure 4.1 – Un chromosome est composé de N *gènes* (i.e. N machines virtuelles). Chaque valeur assignée à un gène est le numéro de la machine physique sur laquelle la machine virtuelle a été allouée.

4.4.2 Opérateurs

Comme évoqué en introduction, un des principes de base d'un algorithme est d'appliquer des opérateurs de façon aléatoire sur les chromosomes d'une population afin d'en former de nouveaux. Les meilleurs individus sont gardés pour faire partie de la génération suivante, puis le processus recommence.

Les opérateurs utilisés, et décrits ci-dessous, sont les trois opérateurs typiques des algorithmes génétiques :

- L'opérateur de *mutation*, appliqué sur un nombre fixé de chromosomes, choisit de façon aléatoire un gène et change sa valeur. Ce changement de valeur du gène concerné signifie que la machine virtuelle représentée par ce gène a été allouée sur une autre machine physique. Ainsi, le nouveau chromosome obtenu représente une nouvelle solution de placement et est intégré à la population courante.
- L'opérateur de *croisement*, également appliqué sur un nombre fixé de chromosomes, intervertit des parties de deux chromosomes et génère ainsi deux nouvelles solutions. Cette opération est effectuée en utilisant deux points de croisement. Un exemple illustré de cet opérateur est donné par la Figure 4.2.
- L'opérateur de *sélection* a pour rôle de réduire le nombre d'individus présents dans la population, temporairement agrandie par l'exécution des deux opérateurs ci-dessus, afin de garder les meilleurs d'entre eux et ainsi redonner à la population sa taille originelle.

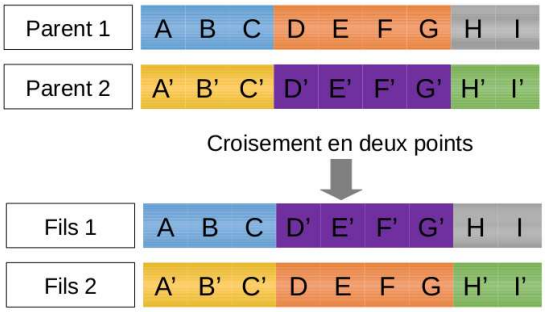


Figure 4.2 – Opérateur de croisement en deux points, appliqué aux chromosomes.

4.4.3 Validité des chromosomes

Un processus important d'un algorithme génétique est de s'assurer que les solutions trouvées, suite à l'application des opérateurs, sont des solutions valides. Autrement dit, les solutions trouvées doivent respecter les contraintes du modèle du problème à résoudre. Pour l'algorithme génétique utilisé ici, cela consiste à vérifier que chaque solution respecte le taux d'utilisation maximum de chaque ressource (CPU et Mémoire). Si la vérification déduit qu'un chromosome ne respecte pas ces contraintes et n'est donc pas valide, alors il est simplement supprimé de la population courante et un nouvel individu est généré.

4.4.4 Critère d'arrêt

La terminaison d'un algorithme génétique peut être décidée de deux manières différentes :

- En définissant un seuil d'amélioration qui permet de comparer le meilleur chromosome de la génération actuelle avec le meilleur chromosome de la génération précédente. Si la différence entre ces deux chromosomes est inférieure au seuil défini, alors le GA est stoppé, considérant que l'amélioration entre deux générations consécutives n'est pas assez importante pour qu'il soit intéressant de continuer le processus.
- Après un nombre fixe de générations

Dans l'algorithme étudié ici, la première solution ne pouvait être utilisée. En effet, comme expliqué dans la section suivante (Section 4.4.5), les chromosomes sont évalués selon une valeur normalisée. Celle-ci dépend entre autre de valeurs moyennes de chacune des métriques (calculées pour tous les individus de la population). Ces moyennes sont forcément différentes entre chaque génération. Ainsi, les valeurs de *fitness* des chromosomes ne sont pas comparables entre générations successives. C'est pourquoi, la seconde solution a donc été adoptée.

4.4.5 Métriques et valeurs de Fitness

Chaque calcul de métrique donne une valeur dans un intervalle intrinsèquement lié à la métrique concernée. Afin de pouvoir calculer de façon correcte la valeur de la fonction objectif, mettant en jeu un ensemble de métriques dont les valeurs ne sont pas comprises dans les mêmes intervalles, une normalisation de chacune de ces valeurs de métrique doit être appliquée. Cela consiste à calculer, pour chaque métrique, une valeur nommée v_{std} normalisée. Ces valeurs sont donc comparables et il est alors possible de les additionner ou les soustraire entre elles afin d'obtenir une valeur normalisée de *fitness*. La méthode de normalisation utilisée ici est la méthode "Centrer-Réduire" dont la formule donne un ensemble de valeurs dont la moyenne est 0 et une variance de 1.

$$v_{std} = \frac{v - \mu}{\sigma} \tag{4.4.1}$$

avec, v la valeur de la métrique à normaliser, μ la moyenne de la métrique sur l'ensemble de la population et σ l'écart-type. Ainsi, la valeur normalisée de *fitness* est égale à une formule linéaire intégrant toutes ces valeurs normalisées. Cela permet donc de comparer de façon correcte et précise chaque chromosome appartenant à une génération en fonction de leur valeur de *fitness*.

La fonction objectif utilisée dans cet algorithme génétique est la suivante :

$$F_{obj} = \alpha_1 E + \alpha_2 RespT + \alpha_3 Rob - \alpha_4 Dyn \tag{4.4.2}$$

avec, α_1, α_2, α_3 and α_4 les coefficients respectifs de l'énergie (E), du temps de réponse (RespT), de la robustesse (Rob) et du dynamisme (Dyn). Ces coefficients peuvent être modifiés (augmentés ou diminués) pour avantager l'optimisation d'une (ou plusieurs) métriques. Dans cette équation 4.4.2 de la fonction objectif, l'énergie, le temps de réponse et la robustesse sont des métriques à minimiser, contrairement à la métrique de dynamisme, qui elle, doit être maximisée.

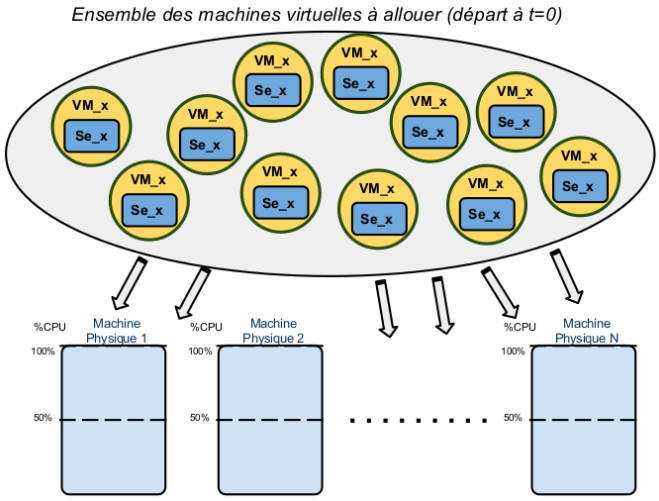


Figure 4.3 – Illustration du problème d'allocation de services élémentaires débutant tous leur exécution à $t = 0$

4.4.6 GA version : Allocation de services élémentaires

Cette première version de l'algorithme génétique met en jeu uniquement une allocation de services élémentaires indépendant, chacun exécuté dans une machine virtuelle dédiée. Chaque machine virtuelle représente donc un service élémentaire dont la date de départ est $t = 0$. Comme illustrée en Figure 4.3 la résolution du problème consiste à trouver un placement de l'ensemble des machines virtuelles sur les machines physiques disponibles tout en optimisant les métriques de QoS présentées en Section 4.2. Il est évident que cette version de l'algorithme génétique résout un problème d'allocation de services très simplifiés par rapport à de réels services *Cloud*. Cependant, cette version d'allocation de services simplifiés permet de focaliser les analyses sur l'optimisation des différentes métriques de QoS sélectionnées. En effet, utiliser cet algorithme génétique comme méta-heuristique de placement permet d'évaluer à la fois la qualité d'optimisation de celui-ci mais également de démontrer l'impact des métriques les unes sur les autres, puis de mettre en avant l'intérêt d'utiliser une approche multi-critères à des fins d'allocation de services.

C'est cette version de l'algorithme génétique qui est utilisée dans la suite des travaux présentés. L'analyse de l'impact de l'optimisation multi-critères est discutée de façon détaillée en Section 4.4.8.

4.4.7 GA version : Allocation de services composés

Une seconde version de l'algorithme génétique utilisant des Service Composés (S_c) a également été explorée. Pour rappel, comme défini en Section 3.2.2, un service composé est un ensemble de services élémentaires (un service élémentaire est exécuté par une et une seule machine virtuelle). Dans un cas général, la topologie d'un service composé est un DAG. Cela permet de définir les relations de dépendances entre les services élémentaires et ainsi représenter l'ordre d'exécution de chacun d'entre eux. Une des principales différences par rapport à la version présentée dans la section précédente, est que l'utilisation de services composés apporte un aspect temporel dans l'exécution des machines virtuelles.

Topologie

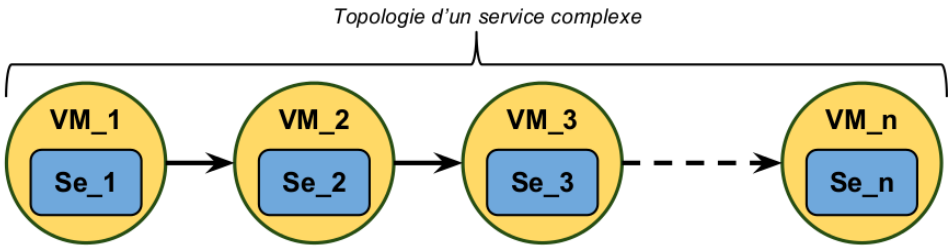


Figure 4.4 – Illustration de la topologie d'un service composé

Pour cette première version de l'algorithme génétique utilisant des services composés, une topologie très simple a été choisie (illustrée en Figure 4.4). Cette topologie permet cependant d'introduire une dépendance entre les services élémentaires d'un service composé entraînant des dates de départ différentes entre chaque service élémentaire ainsi qu'un temps de communication entre deux services élémentaires consécutifs. De plus, une machine virtuelle ne peut appartenir qu'à un seul service composé.

Génération des services composés

L'ensemble des machines virtuelles est utilisé afin de créer un certain nombre de services composés. La taille d'un service composé a été fixée entre 2 et 10 (choisie aléatoirement à sa création), et les machines virtuelles qui le composent sont également choisies aléatoirement. En utilisant la configuration : 400 machines virtuelles / 110 machines physiques, cela amène donc à avoir un nombre de services composés entre 40 et 200, de tailles différentes. Le temps de communication entre deux services élémentaires est fixe (1 seconde).

Calcul des métriques

Malgré ces nombreuses simplifications apportées à la configuration de ce GA, la prise en compte des dépendances entre les dates d'exécution des machines virtuelles, complexifie très nettement le calcul des métriques de QoS considérées. En effet, afin de pouvoir calculer l'énergie totale consommée pour l'exécution de l'ensemble de ces services composés, chaque date de départ et de fin d'exécution doivent être connues. Celles-ci sont variables en fonction des reconfigurations et des services composés auxquels elles appartiennent. L'optimisation de la métrique de consommation d'énergie est donc nettement plus complexe à mettre en place. Concernant la métrique de temps de réponse, celle-ci représente toujours le temps total d'exécution, ici, déterminée par la date de terminaison du dernier service élémentaire du dernier service composé. Son calcul est un peu plus fin car le ralentissement d'une des machines virtuelles d'un service composé doit se répercuter sur l'exécution des machines virtuelles qui s'exécutent après.

Afin de clarifier l'avancement de cette version du GA : tout ce qui a été cité au-dessus est opérationnel, les nouvelles problématiques évoquées ci-après restent à être clarifiées.

L'optimisation des deux autres métriques soulève encore de nouveaux questionnements. En effet, le dynamisme et la robustesse étant en soi des métriques faisant état d'une propriété du système ne dépendant pas du temps, leurs évaluations dans cette version du GA doivent être remises en question. Plusieurs solutions sont donc envisageables :

- Optimiser leur valeur moyenne sur la durée totale d'exécution
- Optimiser leur valeur maximum atteinte
- Optimiser leur valeur de manière à ce qu'elle ne descende jamais en dessous d'un certain seuil

Quelque soit la solution adoptée, cela implique une évaluation des valeurs de ces métriques à chaque démarrage et terminaison de machines virtuelles afin de tenir compte de l'évolution de la charge du système au cours du temps.

Cette version de l'algorithme génétique implique un nombre non négligeable de nouvelles contraintes et difficultés très intéressantes qui soulèvent de nombreuses problématiques d'allocation. De plus, l'utilisation de services composés permet de se rapprocher un peu plus des contraintes d'allocation de services *Cloud* réels. Cependant, de nombreuses questions restent encore ouvertes, et l'étude de cette version de l'algorithme génétique constitue encore une part des travaux de recherche actuellement en cours.

La Section suivante présente une étude du comportement de l'optimisation mutli-objectifs de l'algorithme génétique (version services élémentaires), en comparant les résultats de cinq configurations d'optimisations différentes de ce GA.

4.4.8 Optimisation multi-objectifs par le GA

L'algorithme génétique résout le problème de placement de machine virtuelles avec pour but d'optimiser les métriques intégrées au calcul de sa fonction objectif. Comme expliqué en Section 4.4.5 chaque individu de la population de travail est évalué en fonction de la valeur de celle-ci (valeur de *fitness*). La solution retenue est donc l'individu qui aura reçu le meilleur résultat dans l'ensemble. L'avantage de l'algorithme génétique est qu'il permet assez facilement d'ajouter ou de supprimer des paramètres à prendre en compte pour évaluer une population. De plus, la résolution du problème se fait dans un temps raisonnable dû à la propriété des algorithmes génétiques de converger rapidement vers une solution qu'il estime être la meilleure. L'inconvénient de l'algorithme génétique est qu'il donne un résultat de placement pour une situation de départ donnée. C'est à dire qu'il calcule chacune des métriques pour cette situation à l'instant $t = 0$ et ne tient pas compte de l'aspect temporel. Bien sûr les dates de terminaison de chaque machine virtuelle sont prises en compte dans les calculs des métriques, mais l'optimisation se fait à partir du placement de départ (différent selon les chromosomes) et n'est pas remis en question au fur et à mesure que le nombre de machines virtuelles encore en cours d'exécution diminue.

Cette section met en avant les avantages d'une optimisation mutli-objectif tenant compte de quatre métriques de QoS simultanément par rapport à un placement focalisé sur un seul paramètre. Pour cela, cinq versions de l'algorithme génétique ont été générées, les quatre premières correspondent à l'optimisation d'une métrique unique et la dernière optimise de façon équitable les quatre métriques prises en

compte. Cela amène donc à avoir les cinq GA suivants :

Configuration

Les différents paramètres de l'algorithme génétique sont les suivants :

- Nombre de machines physiques : 110
- Nombre de machines virtuelles : 400
- Nombre d'individus de la population initiale : 1500
- Nombre d'individus de la population de travail : 120
- Nombre de croisements : 90
- Nombre de mutations : 120
- Nombre de générations : 600

Étant donné la complexité du problème d'allocation traité par le GA, il n'est pas rare que des solutions d'allocation qu'il génère ne soient pas valides. C'est à dire que le placement proposé ne respecte pas les contraintes de Mémoire et/ou de CPU. Dans ce cas, le chromosome est rejeté et l'algorithme génétique en génère un nouveau. Si cela se répète trop souvent, le temps de création de la population de départ ainsi que de la population de travail de chaque génération peut très rapidement devenir extrêmement long ou même ne jamais finir (dans le cas où les contraintes de placement sont trop lourdes). Avec 110 machines physiques et 400 machines virtuelles, les contraintes d'allocation sont raisonnables mais ont déjà un impact assez pesant sur la création de la population initiale. En effet, pour trouver 1500 individus valides, le GA génère en moyenne aux alentours de 240000 chromosomes non valides. Avec un tel ratio, entre le nombre chromosomes souhaités pour la population initiale et le nombre d'individus non valides, le temps de création de la population de départ est d'environ 4 secondes. Cela équivaut environ 10% du temps total d'exécution du GA. Le nombre de 1500 a donc été considéré comme un bon compromis entre un nombre d'individus de départ pas trop faible afin d'avoir tout de même une chance que la population de départ soit composée d'individus intéressants et un temps de génération raisonnable. Bien qu'il n'existe pas de paramétrage universel pour l'utilisation d'un algorithme, des valeurs de départ donnant de bons résultats sont connues :

- Une population de travail composée de 30 à 50 individus
- Un taux de croisement entre 70 et 95%
- Un taux de mutation de 1 ou 2%
- Un nombre de générations compris entre 30 et 40

Excepté la valeur du nombre de croisements, les valeurs adoptées pour l'algorithme génétique sont plus élevées que ces valeurs théoriques : une population de travail de 120 individus, 75% de croisement, 83% de mutation et un nombre de 600 générations très nettement supérieur. Ces valeurs ont été choisies pour à la fois avoir un temps de résolution total du GA raisonnable (environ 40 secondes), et une recherche de solution efficace étant donné la complexité du problème d'allocation à résoudre. Suite à de nombreuses

phases d'évaluation de performance de ce GA, il s'est avéré qu'il était plus efficace d'utiliser une taille de population de travail raisonnable et un nombre de générations important. C'est pourquoi le choix d'un nombre de 120 individus, permettant un temps de traitement de chaque génération raisonnable, et un nombre de 600 générations favorisant l'optimisation entre générations, a été fait. Concernant les croisements, un pourcentage correspondant à la valeur théorique a été adopté car le processus de croisement a montré qu'il générait un grand nombre de solutions non valides. Il n'était donc pas utile d'en augmenter son nombre. À l'inverse, l'opérateur de mutation s'est révélé être très efficace pour générer de bonnes solutions. Appliqué au problème étudié ici, cet opérateur correspond à faire migrer une machine virtuelle sur 100 individus différents à chaque génération.

Configurations d'optimisation

L'algorithme génétique a été décliné en 5 versions différentes. Chacune d'entre elle applique une optimisation différente des métriques :

- *GA_Energy* optimise uniquement la métrique d'énergie
- *GA_RespT* optimise uniquement la métrique de temps de réponse
- *GA_Rob* optimise uniquement la métrique de robustesse
- *GA_Dyn* optimise uniquement la métrique de dynamisme
- *GA_All* optimise simultanément et équitablement ces quatre métriques.

Les valeurs des coefficients correspondant à chacune des versions du GA sont résumées en Tableau 4.1.

Nom GA	**Coefficients appliqués aux métriques**			
	Énergie	**Temps de réponse**	**Robustesse**	**Dynamisme**
GA_All	1	1	1	1
GA_Energy	1	0	0	0
GA_RespT	0	1	0	0
GA_Rob	0	0	1	0
GA_Dyn	0	0	0	1

Table 4.1 – Tableau récapitulatif des différentes versions de l'algorithme génétique associées à leurs coefficients d'optimisation de chacune des métriques de QoS.

Les valeurs de coefficients indiquées dans le Tableau 4.1 pour chaque version du GA correspondent aux poids attribués aux coefficients $[\alpha_1 ... \alpha_4]$ utilisés dans le calcul de la fonction objectif. Lorsque qu'une valeur égale à 0 est appliquée au coefficient d'une métrique, alors celle-ci est totalement ignorée. Ainsi, pour les quatre versions mono-optimisation (GA_Energy, GA_RespT, GA_Rob et GA_Dyn), la valeur de 1 permet d'optimiser uniquement la métrique voulue. Une autre valeur, supérieure à 0 et différente de 1 produirait exactement le même effet sur l'optimisation. Pour la version multi-objectifs (GA_All), la valeur de 1 (lorsqu'elle n'est pas de 0) appliquée aux différents coefficients n'a aucune signification en soi. En effet, bien qu'il s'agisse ici d'une optimisation multi-critères concernant des métriques ayant des unités

différentes, une méthode de normalisation (décrite en Section 4.4.5) a été appliquée à chacune d'entre elles. Une fois que le calcul de la fonction objectif utilise des valeurs normalisées de métriques, la valeur absolue de chacun des coefficients n'a alors aucune importance. En effet, c'est uniquement la différence des valeurs appliquées à ceux-ci qui peut avantager une métrique par rapport aux autres. En d'autres termes, une même valeur positive et différente de 1 aurait pu être appliquée à tous les coefficients du GA_All, donnant un poids égal à chaque métrique considérée dans le calcul de la fonction objectif.

Comparaison des résultats d'optimisation

Cette section compare et critique les différents résultats d'optimisation obtenus avec les différentes versions de l'algorithme génétique.

Les Figures suivantes : 4.5, 4.6, 4.7, 4.8 montrent les résultats des exécutions des différents GA décrits ci-dessus, en utilisant un nombre croissant de machines virtuelles à allouer. Cela résulte donc de 8 exécutions de chacune des 5 versions du GA : les 8 exécutions correspondant à un nombre de machines virtuelles à placer, allant de 50 à 400, sur les 110 machines physiques disponibles. Cette variation du nombre de machines virtuelles à allouer permet d'analyser les résultats d'optimisation appliqués à un système peu chargé, moyennement chargé et très chargé. La première figure (Figure 4.5) présente les résultats de la métrique d'énergie, la deuxième (Figure 4.6) présente les résultats de la métrique du temps de réponse, puis les résultats de la métrique de robustesse sont présentés par la Figure 4.7, enfin la Figure 4.8 présente les résultats de la métrique de dynamisme.

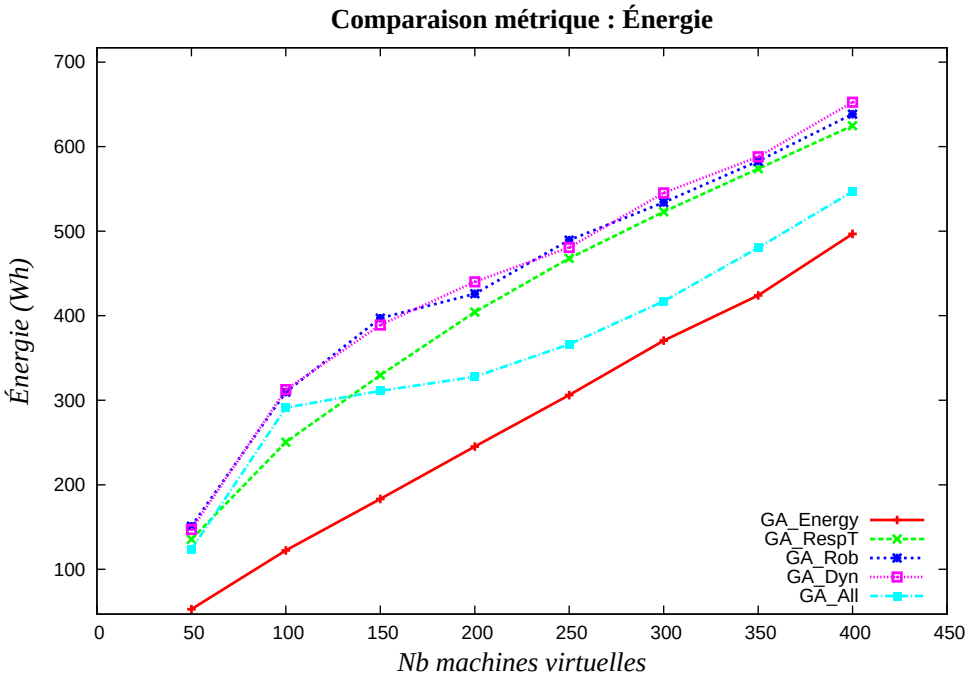


Figure 4.5 – Comparaison des résultats sur la métrique d'énergie

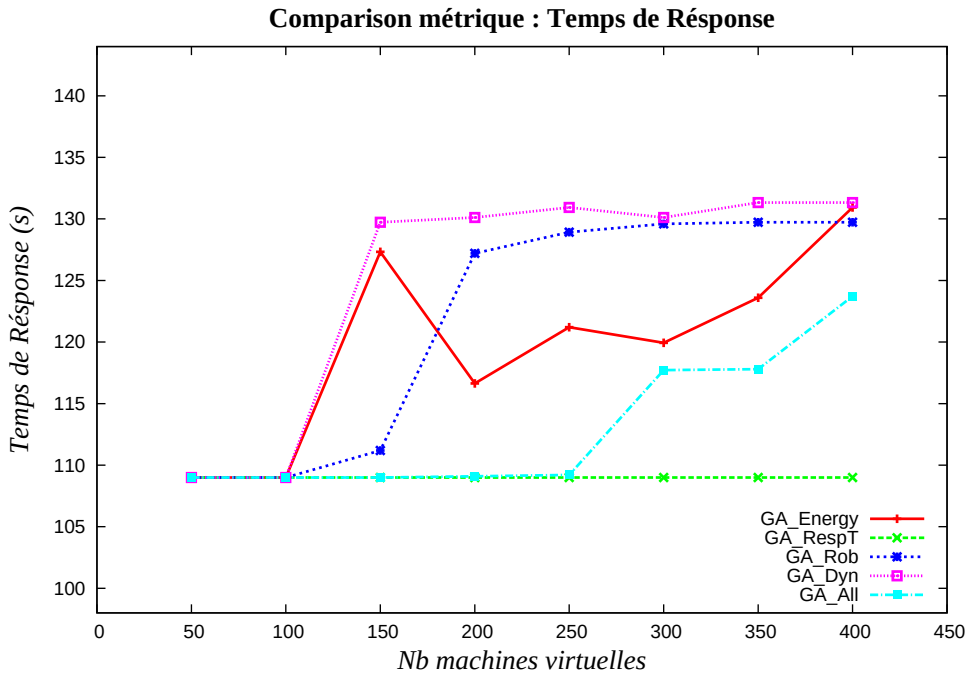


Figure 4.6 – Comparaison des résultats sur la métrique de temps de réponse

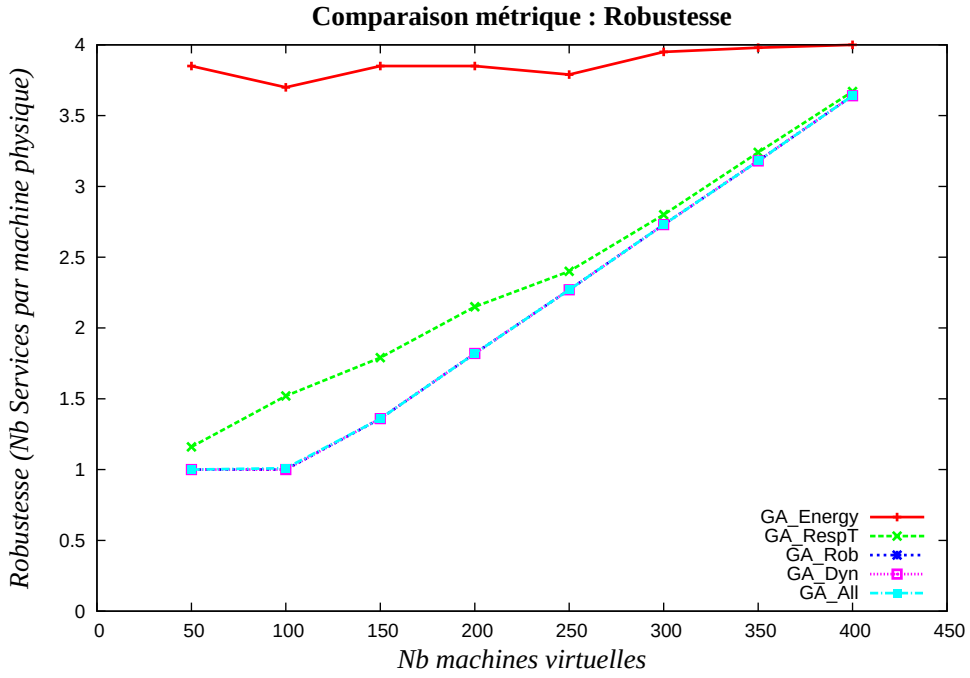


Figure 4.7 – Comparaison des résultats sur la métrique de robustesse

Chacune des figures 4.5 à 4.8 contient cinq courbes. Elles correspondent aux cinq versions (GA_Energy, GA_RespT, GA_Rob, GA_Dyn et GA_All.) de l'algorithme génétique décrites précédemment. Avec, sur l'axe X le nombre de machines virtuelles à allouer, et sur l'axe Y la valeur de la métrique considérée.

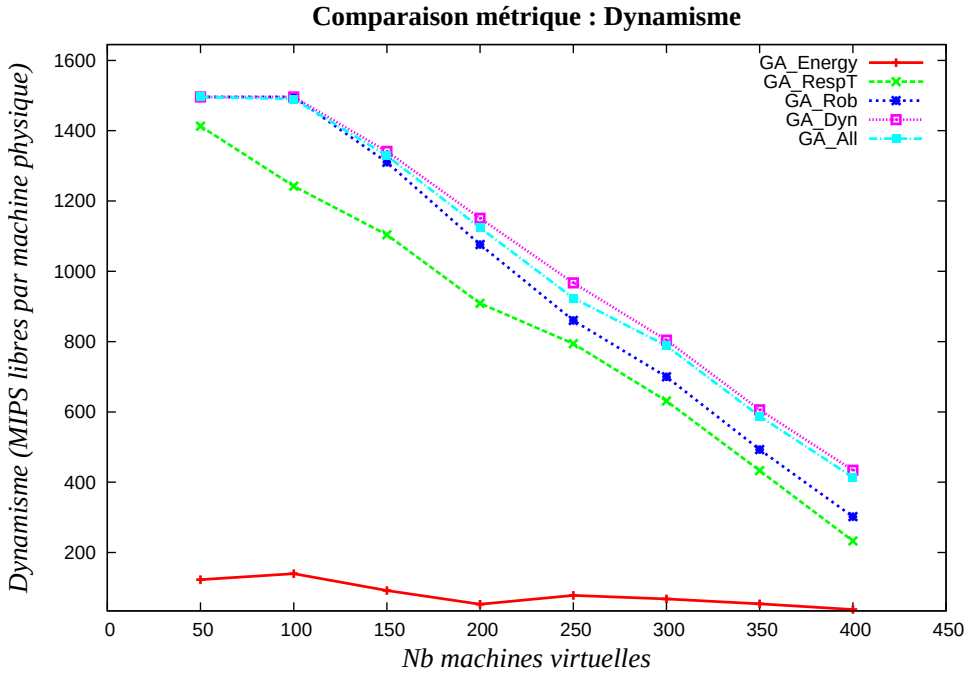


Figure 4.8 – Comparaison des résultats sur la métrique de dynamisme

L'analyse de ces courbes permet de remarquer que sur chacune d'entre elles, la courbe représentant le meilleur résultat correspond à la courbe de la version du GA qui optimise la métrique concernée par la figure. Ce résultat est à la fois logique et rassurant. De plus, cela permet aussi de voir l'écart des autres GA par rapport à la meilleure solution. Les différences les plus remarquables sont données par la version du GA optimisant uniquement la métrique énergie. En effet, si l'on regarde les résultats de ce GA pour la Robustesse ou le Dynamisme, il est indéniable que cette version du GA (GA_Energy) donne de bien moins bons résultats que les autres. Cela renforce aussi l'idée que la consommation énergétique est un paramètre très intéressant à étudier en parallèle de ces deux autres métriques. De plus, outre l'aspect environnemental sous-jacent dans l'optimisation de cette métrique, cela permet d'en déduire également qu'elle est très pertinente dans ce contexte d'optimisation multi-objectifs, en étant parfaitement antagoniste avec la robustesse et le dynamisme.

Ensuite, il convient d'analyser les résultats du *GA_All*, optimisant de façon équitable les quatre métriques. Tout d'abord, on peut remarquer que cette courbe n'est jamais très loin de la meilleure courbe sur chacune des figures. Cela signifie de prime abord que cette version du GA obtient globalement de bons résultats pour chacune d'entre elles. Contrairement aux autres versions du GA étudiées qui dégradent automatiquement une des métriques, cette version semble pouvoir donner d'assez bons résultats et trouver un compromis intéressant de manière à ne défavoriser aucun des paramètres de QoS.

Afin de pouvoir analyser et comparer tous les résultats du *GA_All* par rapport aux autres et donc de mieux se rendre compte de ses performances et des compromis qu'il engendre, une dernière figure (Figure 4.9) est proposée. Sur celle-ci, on retrouve en abscisse le nombre de machines virtuelles à allouer, et l'axe Y représente cette fois la valeur de *fitness* de la fonction objectif. Cette valeur est calculée en sommant

les valeurs normalisées de chaque métrique. La méthode de normalisation appliquée utilise l'intervalle $[min; max]$, correspondant à la plus petite et à la plus grande valeur obtenue par les différentes versions du GA pour toutes les métriques et pour un nombre de machines virtuelles donné. Huit intervalles sont calculés, correspondant aux huit valeurs de l'axe X et les valeurs normalisées sont calculées comme suit :

$$v_{std} = \frac{x - min}{max - min} \tag{4.4.3}$$

avec x la valeur de la métrique. Cette normalisation ramène toutes les valeurs des métriques entre 0 et 1. Ainsi, pour les métriques devant être minimisées, une valeur proche de 0 représente un bon résultat et les valeurs proches de 1 sont elles plus mauvaises. Quand une métrique doit être maximisée, telle celle du dynamisme, l'interprétation de cette valeur est inversée.

Cette méthode de normalisation a été préférée dans cette étude par rapport à la méthode "Centrer-Réduire" car les valeurs de moyennes et d'écart-types utilisées par cette dernière auraient dû être calculées sur un échantillon de cinq valeurs, ce qui est trop peu pour avoir une bonne précision.

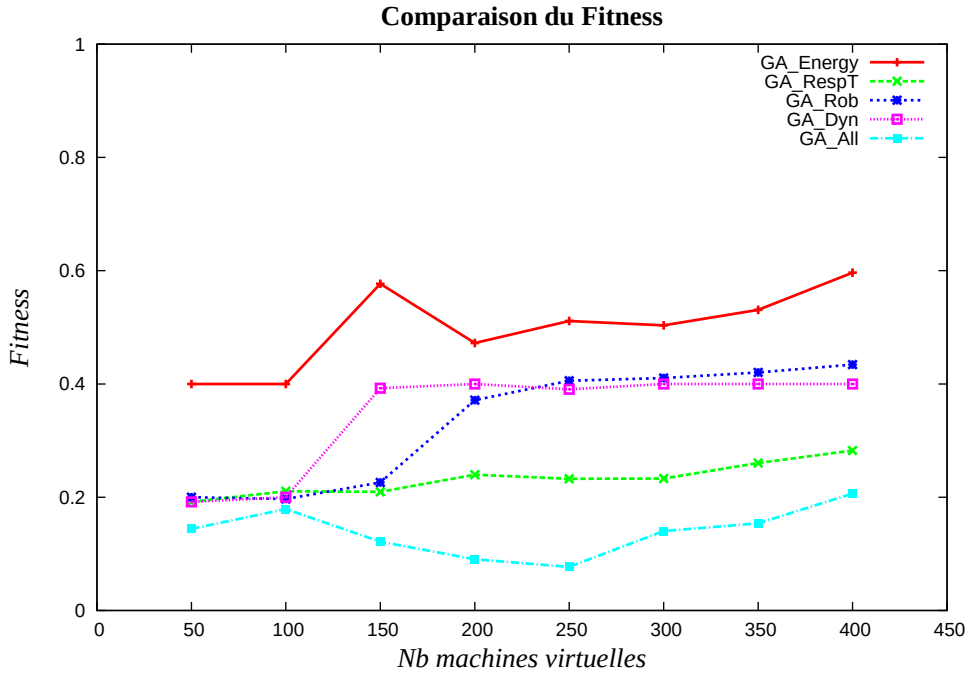


Figure 4.9 – Comparaison des valeurs de *Fitness* entre les 5 versions du GA. La normalisation des valeurs est obtenue en utilisant l'intervalle [min ;max] de chaque métrique pour chaque configuration (i.e. Nombre de machines virtuelles à allouer).

L'analyse de cette comparaison des cinq versions du GA permet de voir que l'approche multi-objectifs, représentée par la courbe GA_All, obtient une valeur de *fitness* meilleure que toutes les autres versions du GA. Cela indique qu'en tenant compte des quatre métriques évaluées, aucun des GA optimisant une seule d'entre elles n'est en mesure d'obtenir un meilleur résultat que la version *GA_All*, quelle que soit la charge du système représentée par l'augmentation progressive du nombre de machines virtuelles à allouer.

Outre une comparaison de différentes versions de l'algorithme génétique, cette analyse a aussi pour but de montrer l'intérêt de l'utilisation d'une optimisation multi-objectifs autant pour un fournisseur de services de *Cloud Computing* que pour ses utilisateurs. En effet, cette approche multi-objectifs permet à la fois d'obtenir de multiples informations quant aux valeurs métriques mesurées en fonction de la charge du système, mais aussi de donner la possibilité au fournisseur de services de prendre des décisions de configurations de son système dans le but de trouver le meilleur compromis possible entre des paramètres de qualité de service antagonistes : Performance versus Robustesse ou Énergie versus Dynamisme, par exemple. Aussi, cette approche permet de choisir un compromis entre les bénéfices favorables au fournisseur de services et la qualité de service offerte à ses utilisateurs.

Par exemple, si le fournisseur de services remarque que la charge de son système est assez faible (peu de machines virtuelles en cours d'utilisation), alors il peut, par exemple, vouloir appliquer une configuration d'allocation de ses machines virtuelles avantageant la robustesse et le temps de réponse afin d'être certain de respecter les valeurs de performance décrites dans le SLA et promises aux utilisateurs, sachant que cela ne nécessitera pas d'avoir un grand nombre de machines physiques allumées et n'engendra donc pas une consommation énergétique élevée. Pour adopter la configuration qui va répondre le mieux à son attente, alors il peut comparer les résultats d'allocation venant d'optimisations multi-objectifs avec des coefficients qu'il aura fixés en fonction de ses besoins par rapport à d'autres solutions qui optimisent seulement un ou deux paramètres (robustesse et de temps de réponse par exemple), puis choisir le compromis qu'il estimera être le meilleur pour son système.

D'autres situations peuvent entraîner le fournisseur de services à choisir une optimisation multi-objectifs équitable entre les critères de QoS qu'il prend en compte, par exemple quand son système est assez ou très chargé. Dans ce cas, les résultats de l'optimisation de l'ensemble des paramètres mesurés peuvent rassurer le fournisseur, en lui fournissant une configuration de son système à la fois assez conservatrice d'un point de vue énergétique tout en limitant, dans une certaine mesure, le risque de violation de valeurs SLA garanties aux utilisateurs. Par exemple, un fournisseur de services peut estimer qu'il a intérêt à opter pour une telle configuration s'il estime qu'il ne dégrade que d'environ 5% certaines métriques, augmentant faiblement le risque de violation de SLA.

C'est donc par cette vision des choses, qu'une approche multi-objectifs peut se révéler être efficace à la fois pour les fournisseurs de services et les utilisateurs suivant le taux d'utilisation de la plate-forme.

Les résultats de l'algorithme génétique analysés ci-dessus démontrent donc qu'une approche multi-critères de QoS *Cloud* peut se révéler intéressante dans différentes situations. Seulement, celle-ci fournit un placement pour une situation figée dont la qualité se dégrade inévitablement au cours du temps, au fur et à mesure que les machines virtuelles terminent leur exécution, ou dans un cas plus général lorsque la charge du système évolue. C'est pourquoi, il est utile de remettre en question l'allocation attribuée à un instant t (dans le passé), en exécutant à nouveau un algorithme de placement tenant compte de l'état actuel du système. Ce procédé est appelé ré-allocation.

4.5 Du placement à l'ordonnancement

Les approches présentées dans ce chapitre résolvent le problème de placement (ou allocation statique) de machines virtuelles pour une situation donnée. C'est à dire, un nombre de machines virtuelles, un nombre

de machines physiques et un ensemble de métriques à optimiser. Le placement obtenu est donc une solution au problème décrit par cette configuration à un instant t. Les solutions proposées font abstraction totale d'une éventuelle évolution du système, en ignorant l'aspect temporel. Il est aisément compréhensible que ces résultats de placement sont efficaces pour des systèmes ayant une durée de vie très brève. En effet, plus la durée de vie d'un système est courte moins il est sujet à de nombreux changements de configuration ou d'utilisation au cours de son existence. À l'inverse, des services s'exécutant pendant de longues heures, semaines ou mois, voient leur taux d'utilisation varier très nettement au cours du temps. Il est évident que dans les *Clouds* l'utilisation des *data centers* des fournisseurs de services varie constamment suivant les différentes périodes de l'année, des saisons, des mois, des jours et encore plus précisément suivant la durée de vie d'un service. Une solution de placement simplement statique ne peut donc pas satisfaire les besoins des fournisseurs de services qui voient l'état de leur système changer constamment. Il est donc nécessaire de revoir le placement effectué à l'instant t, périodiquement ou en fonction d'un événement pré-défini. Ce procédé qui a pour but de redonner un meilleur rendement au système considéré en déplacement les machines virtuelles est appelé : ré-allocation. Ainsi, lorsque le taux d'utilisation diminue, ou au contraire, augmente fortement il peut être intéressant pour les fournisseurs de service de faire appel à un processus de ré-allocation afin de mieux utiliser l'ensemble des machines physiques disponible. Cette étape peut avoir différents buts. Par exemple, si une ré-allocation est lancée alors que le nombre de machines virtuelles a fortement diminué, celui-ci sera plutôt utilisé afin d'obtenir un compromis entre une consolidation de celles-ci favorable à la consommation d'énergie tout en dégradant le moins possible les autres paramètres de QoS. Dans le cas inverse une ré-allocation sera l'occasion d'allouer l'ensemble des nouvelles machines virtuelles tout en déplaçant celles qui étaient déjà en cours d'utilisation. En effet, il sera toujours plus intéressant, d'un point de vue énergétique et de qualité de service à long terme, de remettre en question l'ensemble des allocations des machines virtuelles plutôt que d'entasser les nouvelles sur les machines physiques déjà en cours d'utilisation et ayant assez de capacité libre, puis d'allumer de nouvelles machines physiques afin d'allouer rapidement les nouvelles machines virtuelles qui doivent être démarrées.

Bien sûr, beaucoup de solutions intermédiaires peuvent exister et le choix entre une solution plutôt qu'une autre peut dépendre de nombreux paramètres :

- Dernier instant de ré-allocation
- Charge actuelle du système
- Temps d'utilisation des machines virtuelles en cours de fonctionnement
- Temps d'utilisation prévu des nouvelles machines virtuelles à allouer
- Estimation du ratio entre le nombre de machines physiques disponibles et le nombre de machines virtuelles total après ré-allocation
- Niveau de qualité de service actuel de l'ensemble du système, etc...

Les prises de décision quant au démarrage d'un processus de ré-allocation peuvent donc dépendre de paramètres très variés. De plus, une autre problématique à résoudre est de décider à quel moment il est le plus judicieux de lancer celui-ci. Là encore, bien des situations différentes peuvent se présenter. Par exemple, si le fournisseur de services prévoit que dans l'heure qui suit le taux d'utilisation des ces *data centers* va diminuer de 70% pour un certain laps de temps, il est sûrement préférable d'attendre ce

moment avant de procéder à une ré-allocation des machines virtuelles en cours d'utilisation. Au contraire, si la dernière ré-allocation remonte à plusieurs heures car le taux d'utilisation était plutôt stable et qu'un nombre nombre non négligeable de machines virtuelles a dû être alloué rapidement, alors il n'est sûrement pas judicieux de rester dans une situation d'allocation non optimisée et attendre trop longtemps avant de procéder à une ré-allocation de l'ensemble de celles-ci.

Toutes ces situations assez complexes peuvent aussi poser la question du choix du moment de ré-allocation, c'est à dire comment choisir l'intervalle de ré-allocation le mieux adapté au système concerné.

4.5.1 Intervalle de ré-allocation

La nécessité de relancer un processus de placement pendant l'exécution des services pose le problème de trouver ou de définir un intervalle de temps entre ceux-ci. La première solution est de définir un intervalle fixe entre deux placements. La seconde, décide du moment opportun pour refaire un placement. Celui-ci peut être décidé par exemple en fonction de l'évolution d'un ou plusieurs paramètres de qualité de service, surveillés par les fournisseur de services. La prise de décision peut donc par exemple dépendre d'un taux de dégradation de certains paramètres, mis en évidence par la mesures des métriques associées à ceux-ci et par comparaison à un seuil critique défini à l'avance.

Le choix de l'une de ces solutions soulève aussi le choix entre d'une prise de décision autonome par le système ou un simple envoi d'informations au fournisseur de services qui se réserve la possibilité d'agir ou d'ignorer la prise de décision autonome du système.

4.5.2 Solutions adoptées

Les différentes problématiques liées à la manière dont il est judicieux effectuer les ré-allocations et à la prise de décision quant aux intervalles de ré-allocations, permettent de mettre en avant les enjeux, plus ou moins complexes, que soulève le passage d'un problème d'allocation simple à un problème d'ordonnancement.

Pour transformer le problème de placement présenté dans ce chapitre en problème d'ordonnancement, l'aspect temporel est introduit par l'utilisation d'un simulateur de *Cloud Computing*. L'utilisation de ce simulateur, CloudSim, permet donc de dérouler l'exécution des machines virtuelles et avoir ainsi une idée de l'évolution de l'utilisation des machines physiques au cours du temps.

Dans les simulations étudiées dans ce manuscrit, toutes les machines virtuelles démarrent en même temps ($t = 0$). Seules leurs hétérogénéités en termes de puissance de calcul, de nombre d'instructions à exécuter et des reconfigurations qu'elles subissent en fonction de leur allocation modifient le temps de total d'exécution. Aucune nouvelle machine virtuelle est introduite dans le système après $t = 0$. Une telle configuration entraîne donc une diminution progressive de la charge totale du système (illustrée en Figure 4.10). Cependant, la manière exacte dont la charge système diminue au cours du temps est inconnue. Celle-ci dépend des allocations attribuées par les algorithmes de placement engendrant plus ou moins de reconfigurations de machines virtuelles et des réglages différents de fréquence CPU.

Sachant cela, utiliser un intervalle de ré-allocation fixe semblait être une solution intéressante. En effet, le nombre de machines virtuelles en cours d'exécution diminuant plus ou moins progressivement,

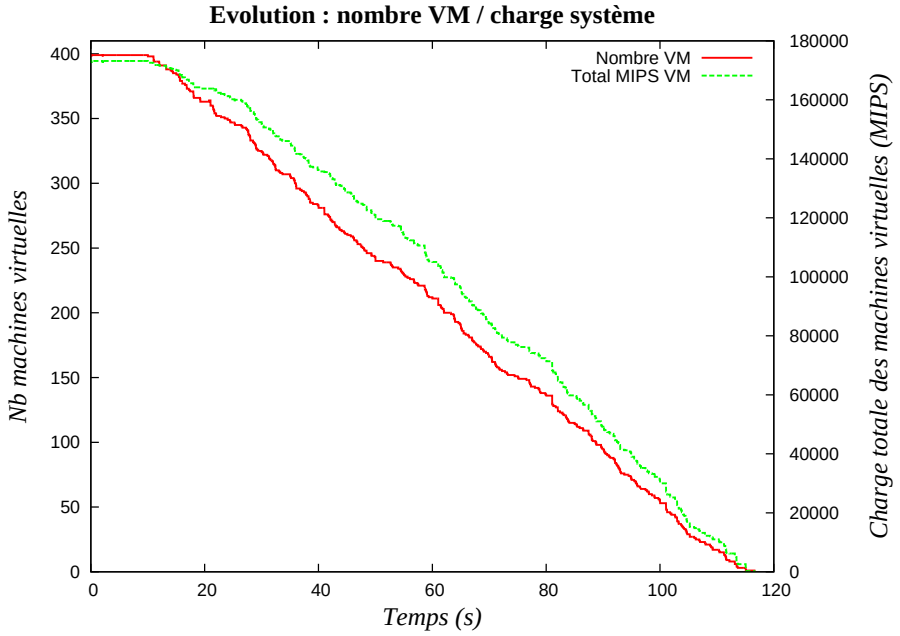


Figure 4.10 – Évolution du nombre de machines virtuelles en cours d'exécution et de la charge système associée (somme des MIPS de l'ensemble des machines virtuelles). L'axe Y de gauche correspond au nombre de machines virtuelles, celui de droite à la charge système (MIPS).

effectuer des ré-allocations à intervalle régulier permet d'analyser l'influence des ré-allocations sur les métriques de qualité de service sélectionnées. Bien sûr, ce choix dépend largement du fait que le nombre de machines virtuelles ne peut subir de grandes variations au cours du temps. Le cas échéant, un tel comportement pourrait avoir une influence franche sur un ou plusieurs paramètres de QoS ce qui aurait poussé à opter pour des déclenchement de ré-allocations en fonction de la valeur de certaines métriques de QoS. Cela afin de pouvoir détecter de brusques variations néfastes pour l'ensemble du système. La solution d'intervalles de ré-allocation fixes adoptée pour les simulations présentées en Chapitre 6 permet également de comparer régulièrement les résultats d'ordonnancement en fonction de l'évolution de la simulations, des comportements des algorithmes gloutons, mais également en fonction des différentes configurations d'optimisation multi-critères utilisées.

Le chapitre suivant présente le simulateur CloudSim et l'intégration des nouvelles fonctionnalités qui ont fait l'objet de plusieurs travaux.

5

CloudSim : simulations énergétiques avec DVFS

Sommaire

Ce chapitre est consacré à l'outil d'évaluation choisi pour cette thèse : le simulateur CloudSim, développé au *CLOUDS Laboratory* de Melbourne en Australie. Ce simulateur a été choisi pour ses qualités de modélisation, de gestion énergétique et ses possibilités d'adaptation et d'évolution. Au cours de ce chapitre, une large partie est consacrée aux évolutions apportées à CloudSim, afin d'y intégrer des fonctionnalités indispensables, et ainsi de faire évoluer ses caractéristiques dans le but d'obtenir un simulateur permettant d'exécuter des simulations énergétiquement pertinentes et précises. Le DVFS, le principe de reconfiguration de machines virtuelles, la gestion dynamique des évènements ainsi que l'intégration de nouveaux modèles énergétiques ont été conçus au sein de CloudSim afin de rendre ce simulateur complet du point du vue de l'intégration et de l'utilisation des outils de gestion de consommation énergétique. Tout cela afin de pouvoir mener les phases d'évaluation présentées dans le chapitre suivant avec toutes les fonctionnalités nécessaires.

Ainsi, les différentes sections de ce chapitre présentent tout d'abord de manière détaillée l'architecture et les caractéristiques de base de CloudSim, puis une description précise des évolutions qui lui ont été apportées. Enfin les différentes phases de validation de ces nouvelles fonctionnalités sont détaillées par l'utilisation de plusieurs cas d'utilisation.

5.1 Le simulateur CloudSim

CloudSim, projet développé par le CLOUDS Laboratory de Melbourne en Australie, est un outil de simulation extensible permettant la modélisation et la simulation d'environnement de systèmes de type *Cloud Computing* de niveau IaaS. Cette section d'introduction aux fonctionnalités de CloudSim est largement inspirée de l'article présentant en détails ce simulateur [24].

5.1.1 Architecture générale

Illustrée en Figure 5.1, son architecture permet la modélisation des différents composants d'un *Cloud* : *Data Center*, machines physiques, machines virtuelles etc... La gestion de toutes ces ressources est gérée par la couche *Cloud Services*, gérant l'approvisionnement des machines virtuelles, l'utilisation des CPUs, de la mémoire RAM, de la capacité de stockage et de la bande passante des machines physiques. Dans CloudSim, les tâches à exécuter sont assimilées à des *cloudlets* qui sont alloués aux machines virtuelles (couches *VM Services* et *User Interface Structure*). La configuration de chacun des composants doit être gérée par l'utilisateur (couche *Simulation Specification*) par l'intermédiaire d'un fichier de configuration de simulation. De nombreux paramètres peuvent y être spécifiés :

- Nombre de *Data Center*, machines physiques, machines virtuelles, tâches (*cloudlets*)
- Caractéristiques des *Data Center* : architecture, système d'exploitation, hyperviseur, coûts de fonctionnement
- Caractéristiques des machines physiques : nombre de CPUs, capacité des CPUs, capacité mémoire, capacité de stockage, bande passante
- Caractéristiques des machines virtuelles : capacité CPUs, nombre de CPUs, capacité mémoire, capacité de stockage
- Caractéristiques des *cloudlets* : nombre d'instructions à exécuter, taille des fichiers d'entrée et de sortie

5.1.2 Modélisation du Cloud

CloudSim est dédié à la simulation de services *Cloud* au niveau Infrastructure (IaaS). Pour ce faire, les simulations sont basées sur la gestion des machines physiques qui composent le ou les *Data Center*. À ces machines physiques sont allouées des machines virtuelles selon des politiques de placement qui peuvent être aisément modifiées. Aussi, le cycle de vie des machines virtuelles (création, allocation, migration éventuelle et destruction) détermine le démarrage, le déroulement et la terminaison des simulations.

5.1.3 Modélisation des machines virtuelles et des Cloudlets

La couche de virtualisation gère l'exécution des machines virtuelles sur les machines physiques et la façon dont elles exécutent les *cloudlets*. En effet, CloudSim propose deux modes d'exécution : temps partagé et espace partagé pour l'exécution des machines virtuelles sur les machines physiques et également pour les *cloudlets* au sein des machines virtuelles. Ce qui permet donc quatre configurations d'exécution différentes. L'exécution des machines virtuelles est également gérée par l'instanciation de *brokers*. Un *broker* peut contenir une ou plusieurs machines virtuelles. La fin de son cycle de vie est déterminé par la durée de

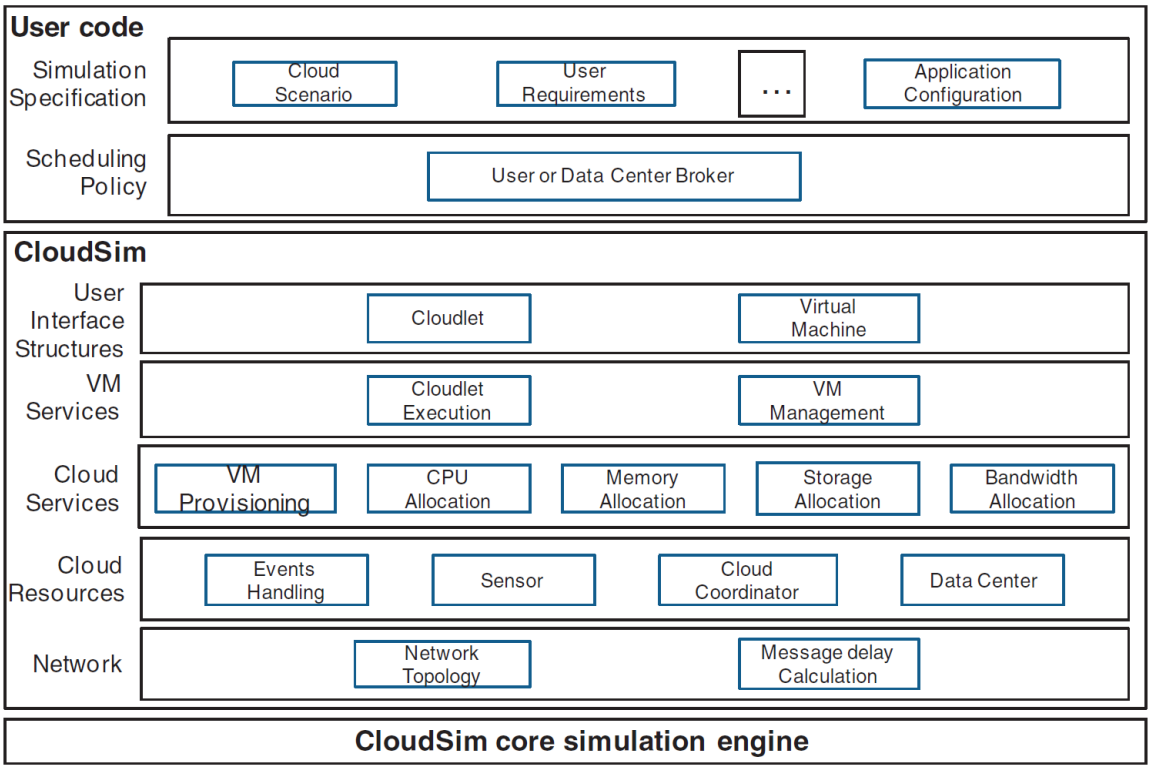


Figure 5.1 – Représentation de l'architecture de CloudSim (présentée dans [24]).

vie de chacune des machines virtuelles qui le composent. Ainsi, un *broker* termine son exécution seulement lorsque toutes les machines virtuelles qu'il contient ont également fini leur exécution.

5.1.4 Politiques de placement et de migration

Des méthodes génériques de placement sont également implémentées ce qui permet à l'utilisateur d'intégrer facilement de nouvelles politiques en fonction du type d'études qu'il souhaite mener. Des algorithmes personnalisés de placement de machines virtuelles peuvent être ainsi intégrés en plus des algorithmes de base présents dans le code source du simulateur. De la même manière des classes abstraites peuvent être aisément étendues pour permettre à l'utilisateur d'implémenter ses propres algorithmes de décision de migration de machines virtuelles.

5.1.5 Consommation énergétique

CloudSim intègre aussi des fonctionnalités permettant de gérer et de calculer la consommation énergétique des *Data Center* et peut être ainsi utilisé pour mener des études sur les problèmes de consommation d'énergie dans le domaine du *Cloud Computing*. En effet, plusieurs outils permettant de gérer la consommation énergétique des machines ou des *Data Center* en général sont disponibles :

- Extinction et/ou allumage des machines physiques

- Modèle énergétique des machines physiques
- Processus de migration de machines virtuelles

Par tous ces aspects, CloudSim est un simulateur solide, dont la configuration est bien faite, et assez facilement extensible par de nouveaux modèles énergétiques et de nouvelles politiques de placement et/ou de migration.

La section suivante détaille comment l'intégration du DVFS a été réalisée afin de compléter les outils de gestion de consommation énergétique de ce simulateur. La version de CloudSim utilisée dans ce chapitre, intégrant le DVFS, est disponible sur le site internet du CLOUDS Laboratory de Melbourne [1].

5.2 Intégration du DVFS

Comme présenté en Section 2.5, le simulateur CloudSim adopté pour cette thèse, contenait déjà plusieurs outils indispensables pour pouvoir effectuer des simulations énergétiques intéressantes. Seul le DVFS ne faisait pas partie des outils disponibles pour étudier de façon plus précise les problèmes de consommation énergétique des machines physiques. C'est pourquoi l'implémentation de cet outil a constitué une étape importante de ces travaux pour ainsi pouvoir exécuter des simulations énergétiques précises. Celle-ci a nécessité tout d'abord une analyse des différentes modes de fonctionnement du DVFS dans le noyau Linux (présenté en Section 2.3.1, mais également une étape de compréhension du code source du simulateur afin de pouvoir intégrer cet outil, nécessitant une granularité (ou "pas de simulation") très fine, de façon rigoureuse. De plus, les autres fonctionnalités qui ont dû être intégrées afin de répondre aux exigences qu'on engendrées le DVFS sont également présentées.

Outre la méthodologie à adopter, l'intégration du DVFS au sein de CloudSim représente une avancée certaine en matière de simulations énergétiques. En effet, l'utilisation du DVFS constitue un vrai domaine de recherche et est utilisé dans de nombreux travaux [126, 73, 141, 80, 86] pour différents types d'applications. La mise en place d'expérimentations sur de réelles plates-formes nécessitent très souvent une lourde étape de configuration avant même de pouvoir lancer les premières exécutions. Cela est d'autant plus vrai pour les expérimentations sur un *Cloud* : qu'il soit publique ou privé la configuration de l'environnement et la mise en place du lancement des expérimentations représentent une charge de travail non négligeable et ne permet pas toujours d'effectuer toutes les mesures voulues en fonction des équipements installés sur les plates-formes. L'approche par simulation peut donc s'avérer être indispensable dans certaines situations. Avec une intégration du DVFS tel qu'il est utilisé sur les plates-formes réelles au sein d'un simulateur de *Cloud*, cela permet de procéder à des travaux de recherche et des études énergétiques de manière rigoureuse et supprimant en grande partie le temps de préparation. De plus, les résultats obtenus par simulation peuvent servir de base pour la configuration du DVFS sur les plates-formes réelles.

La section suivante présente le détail l'implémentation de ce nouveau *package*.

1. www.cloudbus.org/cloudsim/

5.2.1 Description de l'implémentation

Les différents classes composant l'implémentation du DVFS ont été insérées dans un nouveau *package*, appelé *DVFS*, afin d'intégrer cette implémentation proprement au cœur du simulateur. Dans ce nouveau *package* se trouve les cinq classes, illustrées en figure 5.2, correspondant aux cinq modes de fonctionnement comme dans le noyau Linux. Chaque classe représente chacun des gouverneurs des différents modes du DFVS. Leur rôle est de déterminer à quel moment la fréquence du ou des CPUs doit être modifiée. Cette prise de décision est directement liée aux comportements respectifs de chaque gouverneur. Dans le simulateur, un changement de fréquence affecte directement la capacité de calcul d'un CPU dont l'unité est le MIPS (**M**illions d'**I**nstructions **P**ar **S**econde).

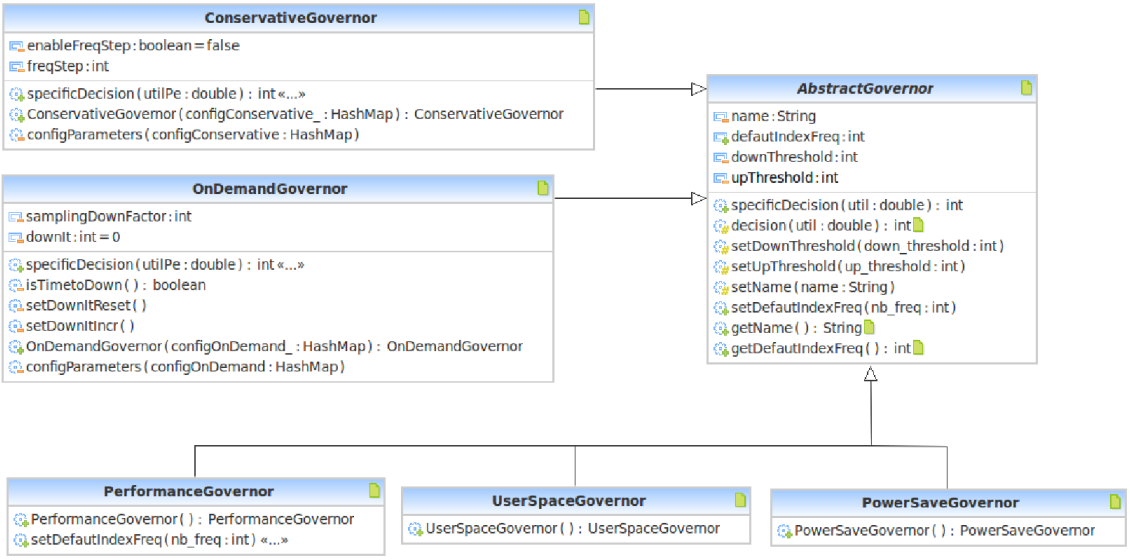


Figure 5.2 – Diagramme UML du *package* DVFS dans CloudSim

L'utilisation du DVFS permet de gérer les performances du CPU et donc des machines physiques. Au cours des simulations, les performances des machines physiques sont donc très régulièrement affectées par ses changements de fréquence CPU, ce qui se répercute sur l'exécution des machines virtuelles qui leur sont allouées. Cela implique donc également de modifier la façon dont le simulateur s'occupe du placement et de la gestion des capacités de calcul des machines virtuelles. Différentes situations nécessitant cette modification de gestion de capacité CPU de machines virtuelles sont illustrées par les exemples suivants :

Exemple 5.1 :

Si le système décide de réduire la fréquence du CPU d'une machine physique, la somme des capacités de toutes les machines virtuelles, en cours d'exécution sur cette machine physique, pourrait temporairement dépasser sa capacité maximale. Dans ce cas, la capacité CPU allouée à chaque machine virtuelle doit être ajustée (revue à la baisse, dans ce cas) proportionnellement à la capacité temporaire de la machine physique, dûe à la nouvelle fréquence CPU utilisée. Ce mécanisme est appelé "reconfiguration de machine virtuelle".

Exemple 5.2 :

Une situation similaire peut se produire lorsqu'une ou plusieurs nouvelles machines virtuelles doivent être allouées à une même machine physique. Dans ce cas, si la somme des capacités des machines virtuelles fonctionnant déjà sur cette machine physique plus les capacités des nouvelles machines virtuelles créées dépasse la capacité totale de la machine physique concernée, alors la capacité CPU de toutes les machines virtuelles doit également subir un ajustement afin que toutes les machines virtuelles puissent continuer à s'exécuter correctement. Évidemment ce procédé affecte temporairement la performance des machines virtuelles concernées.

Exemple 5.3 :

Inversement aux deux exemples précédents, quand une partie de la capacité d'une machine physique est libérée suite à la terminaison d'une ou plusieurs machines virtuelles, alors les machines virtuelles encore en cours d'exécution, ayant subi auparavant une réduction de leur capacité, peuvent récupérer une partie voir la totalité de leur capacité initiale en fonction de l'espace disponible sur la machine physique. Cela arrive dans deux cas :

- Lorsqu'une machine virtuelle a terminé son exécution : alors le pourcentage CPU qu'elle utilisait est libéré.
- Lorsque la fréquence CPU d'une machine physique est augmentée, cela induit donc un surplus de capacité disponible sur la machine physique.

Dans ces deux cas, les capacités de toutes les machines virtuelles encore en cours d'exécution peuvent être augmentées, cela aussi en fonction du pourcentage CPU libre sur la machine physique concernée et en tenant également compte des capacités maximums attribuables à chaque machine virtuelle.

De plus, l'implémentation de la reconfiguration de machines virtuelles et du DVFS, à nécessité d'implémenter d'autres modifications dans le simulateur, notamment au niveau de la gestion des évènements effectués par le noyau de simulateur. En effet, le mode *PowerSave* du DVFS fait fonctionner le processeur à sa fréquence minimum ce qui peut induire un retard conséquent dans l'exécution des machines virtuelles. Dans ces conditions, l'évolution des évènements séquentiels peut se voir perturbé. En effet, si l'exécution d'une ou plusieurs machines virtuelles prend du retard suite aux variations de leurs capacités CPU alors les évènements devant démarrer à la suite des terminaisons ces machines virtuelles doivent inévitablement prendre en compte le retard engendré. Dans ce cas, une description statique (à date fixe) des évènements (Figure 5.3a) n'est plus valide car elle ne répercute pas ce délai d'exécution (illustré sur la Figure 5.3b). Ainsi, une nouvelle fonctionnalité a été ajoutée à CloudSim pour permettre la création d'évènements à la fin de l'exécution d'un *broker*. Cela permet d'attendre la fin de vie de toutes les machines virtuelles, et donc de leur retard d'exécution potentiel, contenues dans le *broker* avant de déclencher le démarrage d'un nouveau *broker*. Pour ce faire, la possibilité d'inclusion d'un *post-event* a été implémentée lors de la création d'un *broker*. Ainsi, un *broker* contenant un *post-event* déclenchera l'exécution d'un nouveau *broker* uniquement à la fin de sa propre exécution (Figure 5.3c).

La Figure 5.3a montre un exemple de base de déroulement d'évènements dans CloudSim, utilisant une description statique des dates de départs des évènements et le mode DVFS *Performance* (fréquence CPU maximum utilisée, donc absence de retard d'exécution dans ce cas). La Figure 5.3b montre un exemple

de déroulements d'évènements, toujours en utilisant une description statique des dates de départ, mais en utilisant le mode *PowerSave* appliquant par exemple une fréquence CPU deux fois plus lente que la fréquence maximum. L'utilisation de ce mode va donc, selon les cas, impliquer un délai plus ou moins conséquent de l'exécution des différents *brokers*. Dans cette situation, il est facile de remarquer que la séquence d'exécution des *brokers* n'est plus correcte, car les déclarations statiques des dates de départs des *brokers* ne tiennent pas compte du retard engendré, ce qui entraîne une superposition des exécutions non souhaitée.

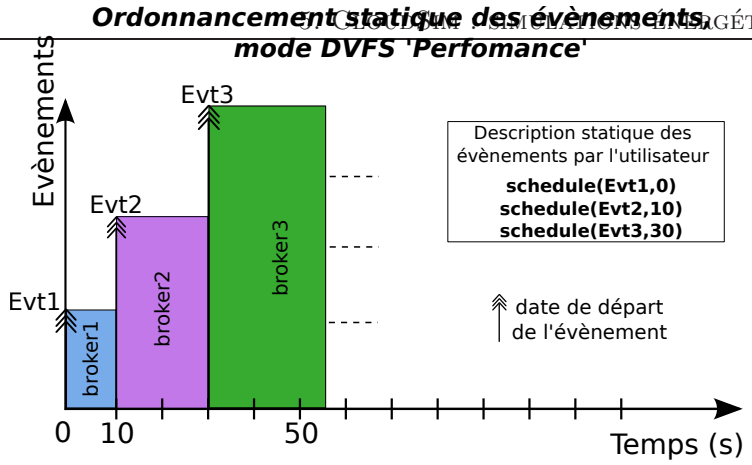
La dernière figure (Figure 5.3c) illustre donc l'utilisation de *brokers* contenant des *post-event* permettant un déclenchement dynamique des évènements. Cette nouvelle fonctionnalité ajoutée dans CloudSim permet ainsi de lier le démarrage d'un *broker* avec la terminaison d'un autre. La date de début d'exécution d'un *brokers* étant ainsi uniquement liée à la terminaison de celui qui le précède, si ce dernier est ralenti (dû par exemple à une fréquence CPU basse) alors cela permet au *broker* suivant de décaler son démarrage afin de respecter le délai généré, supprimant ainsi tout risque de superpositions d'exécutions.

Cette modification autorise d'utiliser n'importe quel mode du DVFS, ralentissant ou non l'exécution des machines virtuelles, sans fausser les résultats des simulations. La déclaration de départ des évènements se fait donc ainsi :

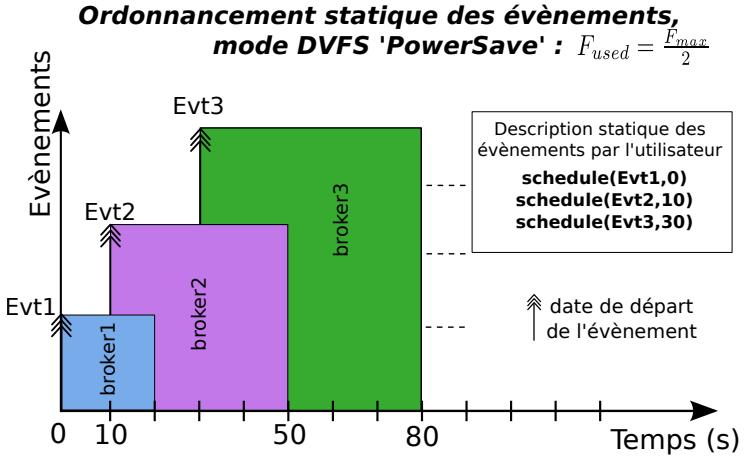
- Les évènements qui doivent démarrer à dates fixes sont déclarés de façon statique, comme le premier évènement de la simulation qui débute à $t = 0$.
- Tous les autres *brokers* qui doivent s'exécuter les uns à la suite des autres sont déclarés avec un *post-event* permettant de déclencher dynamiquement le démarrage des nouveaux *broker* dès leurs terminaisons.

Le dernier changement effectué dans le simulateur a été d'intégrer de nouveaux modèles énergétiques. En effet, les modèles énergétiques dépendent directement des caractéristiques de fréquences et de puissances des machines physiques que l'on veut modéliser. Les modèles utilisés dans ce chapitre sont présentés par les tableaux 5.2 et 5.6.

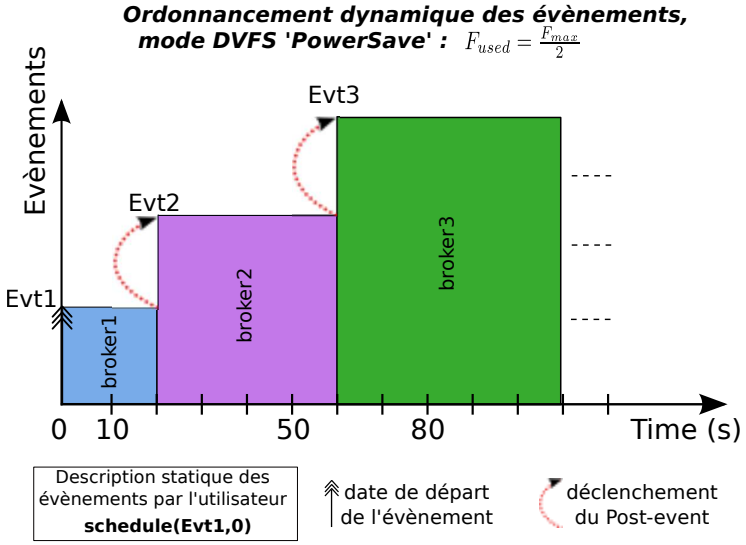
La section suivante présente les phases de validation des nouvelles fonctionnalités apportées à CloudSim par l'intermédiaire de différents cas d'utilisation.



(a) Déroulement des évènements déclarés de façon statique.



(b) Déroulement des évènements déclarés de façon statique en utilisant le mode *PowerSave*.



(c) Déclenchement dynamique des évènements en utilisant le mode *PowerSave*.

FIGURE 5.3 – Illustration du ralentissement généré par l'utilisation du mode *PowerSave* nécessitant une gestion dynamique des évènements.

5.3 Validation

Cette section propose trois cas d'utilisation des nouvelles fonctionnalités intégrées à CloudSim. En effet, il est important de commencer par un cas d'utilisation simple mais parfaitement défini, puis de continuer cette phase de validation par des cas d'utilisation ayant des comportements très différents et plus complexes.Tout d'abord une comparaison du comportement du simulateur par rapport à une simple machine physique est détaillée. Cette première phase de validation, bien qu'assez simpliste car mettant en jeu une seule machine physique, permet de présenter toute la méthodologie adoptée afin d'être certain de la précision de l'approche proposée et donc des comparaisons permettant de juger le fonctionnement des nouvelles fonctionnalités au sein du simulateur. Le deuxième cas d'utilisation utilise une exécution de graphe de tâches (en anglais DAG : *Direct Acyclic Graph*). Cette étape permet d'utiliser le DVFS dans une application ayant un comportement bien spécifique et ainsi de montrer son efficacité avec une configuration de simulation bien différente. Le dernier cas d'utilisation est le plus complexe. Une application distribuée d'électromagnétisme est utilisée sur une plate-forme réelle intégrant le DVFS (Grid'5000 [60, 25]), puis modélisée au sein de CloudSim afin de reproduire son comportement par simulation. Les différents résultats sont également comparés à un modèle analytique dédié à cette application d'électromagnétisme.

Toutes les comparaisons effectuées dans cette section mettent en jeu différents modes de fonctionnement du DVFS, et les résultats obtenus sont confrontés en terme de temps d'exécution et de consommation énergétique.

5.3.1 Machine physique unique

Cette section décrit la phase de validation de l'implémentation du DVFS dans CloudSim à l'aide d'un simple scénario de test. Ce scénario, dédié à la validation du bon comportement du DVFS dans le simulateur (prise de décision de changement de fréquence représentant le comportement réel) à été défini tel un *benchmark* typique, avec différents changements de charge CPU balayant toute la plage d'utilisation du CPU intégrant des changements brusques de charge afin de d'évaluer la réactivé du DVFS dans le simulateur.

L'idée directrice de ce processus de validation est donc d'exécuter une séquence d'instructions sur une machine physique réelle, sur laquelle le DVFS a été activé, et de mesurer la consommation énergétique générée ainsi que le temps d'exécution de ce scénario. Cela en exécutant plusieurs expérimentations utilisant les différents modes de fonctionnement du DVFS. Ensuite, il s'agit de simuler avec précision les mêmes expériences dans CloudSim en utilisant les mêmes configurations de DVFS. La comparaison des résultats des expérimentations et des simulations, permet ainsi d'évaluer la qualité et la précision du fonctionnement du DVFS dans le simulateur. Pour cela, le temps d'exécution total (en seconde) et la valeur de consommation d'énergie totale (en Wh) sont prises en compte. Cette étape de validation se doit d'être abordée de manière très précise dûe à la très fine granularité du fonctionnement du DVFS (prise de décisions très rapides et très fréquentes). C'est pourquoi la méthodologie adoptée est présentée de façon détaillée dans les sections suivantes.

Cadre expérimental

Toutes les expérimentations ont été conduites sur une machine physique standard (nommée *HOST* dans les explications qui suivent), équipée d'un processeur *Intel (R) Core (TM) 2 Quad CPU Q6700@2.66GHz* avec 4Go de mémoire RAM, fonctionnant sous *Ubuntu 10.4 LTS (Noyau Linux 2.6.32)*. Toutes les mesures de puissance ont été réalisées à l'aide d'un wattmètre bluetooth, directement branché entre le secteur et la prise d'alimentation de cette machine, permettant de relever la puissance délivrée par la machine à chaque seconde. Ainsi, cela a permis d'avoir une mesure de puissance précise, puis de calculer la consommation d'énergie totale à chaque exécution.

Étalonnage de la consommation d'énergie

Afin de pouvoir calibrer le calcul de la consommation d'énergie dans CloudSim, il était tout d'abord nécessaire de connaître les valeurs des fréquences admises par le CPU de cette machine physique, récapitulées dans le Tableau 5.1. Dans CloudSim, les fréquences CPU sont assimilées à des capacités de traitement en MIPS, elles ont donc été calculées proportionnellement aux fréquences utilisables sur la machine physique *HOST*. Ensuite, les valeurs de puissance délivrées respectivement à 0% (**Pidle**) et 100% (**Pfull**) d'utilisation CPU ont été mesurées pour chaque fréquence possible. Ces valeurs sont exposées dans le Tableau 5.2 et ont été insérées dans le simulateur afin de créer un modèle énergétique équivalent aux puissances délivrées par la machine physique réelle *HOST*.

Fréquences					
***HOST* (GHz)**	1.60	1.867	2.133	2.40	2.67
%∗$\mathbf{F_{max}}$	59.925	69.93	79.89	89.89	100
CloudSim (MIPS)	1498	1748	1997	2247	2500

Table 5.1 – Fréquences disponibles sur la machine physique *HOST* (en GHz), et celles insérées dans le modèle énergétique de CloudSim (en MIPS). La deuxième ligne de ce tableau exprime en pourcentage chaque fréquence par rapport à la fréquence maximale (F_{max}) de 2.67 GHz.

Charge CPU	**Fréquences (GHz)**				
	1.60	**1.867**	**2.113**	**2.40**	**2.67**
0% (Pidle)	82.7	82.85	82.95	83.10	83.25
100% (Pfull)	88.77	92.00	95.5	99.45	103.0

Table 5.2 – Puissances (en Watt) délivrées par la machine physique *HOST* à 0% and 100% de charge CPU pour chacune des fréquences.

Méthodologie

Le principe est de réaliser une expérience qui implique de nombreux changements de fréquence pour tester le comportement de DVFS, mais aussi des phases de charge constantes (hautes ou basses) afin d'utiliser le modèle de puissance sur toute sa gamme. Dans ce scénario, la valeur maximale de la charge CPU

a été établie à 96% : valeur suffisante pour déclencher des changements de fréquences lors de l'utilisation des modes dynamiques du DVFS (pour rappel : le seuil par défaut du mode *OnDemand* est de 95% d'utilisation CPU).

Le scénario établi se décompose de plusieurs phases, illustrées en Figure 5.5, et se déroule ainsi :

- (A) Montée en charge régulière de 0% à 96%
- (B) Pleine charge à 96%
- (C) Descente linéaire pour atteindre 50% de charge
- (D) Pic de charge à 80%
- (E) Descente linéaire jusqu'à 30%
- (F) Pic de charge à 96%
- (G) Charge constante de 30%

Afin d'exécuter l'application de test sur la machine physique *HOST*, ce scénario a tout simplement été implémenté en langage C. La difficulté soulevée par cette implémentation est de générer exactement la charge CPU moyenne voulue à tout instant du scénario. Pour ce faire, la charge CPU moyenne est générée en utilisant une boucle alternant une phase de calcul intensif puis une phase de sommeil du CPU, comme illustré en Figure 5.4. Lorsque le DVFS est actif, la charge CPU est contrôlée chaque *sampling rate*, correspondant à un très court intervalle de temps de 10ms. Afin d'être sûr de la précision et de la véracité de cette expérimentation, les prises de décisions des différents *gouverneurs* devaient être exactement liées à la génération et aux variations de charge CPU dépendantes de la boucle de calcul. C'est pourquoi, chaque itération de cette boucle doit être exactement égale à la valeur du *sampling rate*.

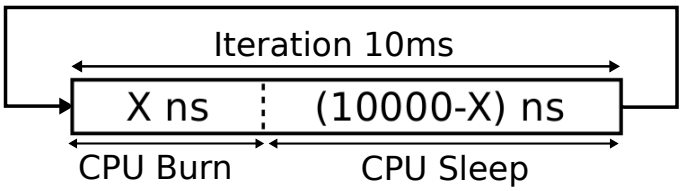


Figure 5.4 – Décomposition de la boucle génératrice de charge en langage C.

Ce scénario est modélisé dans CloudSim en créant un ensemble de *brokers*, chacun contenant une ou plusieurs machines virtuelles dans lesquelles les *cloudlets* sont exécutées. Comme expliqué dans la Section 5.1.3, un *broker* peut contenir un *post-event* permettant alors de démarrer de nouvelles machines virtuelles lorsque celles de ce *broker* se sont terminées. En définissant plusieurs types de machines virtuelles (capacité en MIPS différentes) et plusieurs types de *cloudlets* (nombre d'instructions à exécuter différents), il est alors possible de gérer précisément la montée en charge du CPU de la machine physique considérée. La capacité CPU de la machine virtuelle définit le pourcentage d'augmentation de charge CPU (en admettant que la machine virtuelle soit utilisée à 100% de sa capacité), et le nombre d'instructions des *cloudlets* détermine le temps d'exécution des machines virtuelles auxquelles elles sont attribuées. De cette manière, le scénario est reproduit dans CloudSim par un ensemble de *brokers* lancés les uns à la

	VM/Cloudlet			
	C_1	C_2	C_3	C_4
Charge CPU (%)	10	10	2	2
Durée (s)	1.5	0.8	7.5	4

Table 5.3 – Exemples de charge CPU et de durée d'exécution pour chaque couple $[VM_x, Cloudlet_x]$ définit en utilisant la $Host_1$ ayant une capacité de 1000 MIPS.

suite des autres qui eux mêmes déclenchent l'exécution de machines virtuelles et de *cloudlets*. Enfin, afin d'avoir exactement le même intervalle de temps entre deux prises de décision des gouverneurs des modes du DVFS, le pas de simulation est fixé à $\frac{1}{100}$ seconde.

Afin de mieux comprendre comment la charge CPU dans CloudSim est gérée par la création de *brokers*, machines virtuelles et *cloudlets*, voici un exemple simple mettant en jeu : 1 machine physique, 2 types de machines virtuelles et 2 types de *cloudlets* :

Exemple 5.4 : Couples de machines virtuelles/cloudlets

- $Host_1$: 1000 MIPS
- VM_1 : 100 MIPS , $Cloudlet_1$: 150 Millions of Instructions
- VM_2 : 20 MIPS , $Cloudlet_2$: 80 Millions of Instructions

En considérant la machine physique, les deux machines virtuelles et *cloudlets* ci-dessous, 4 combinaisons de machines virtuelles/*cloudlets* sont possibles :

- $C_1 = [VM_1, Cloudlet_1]$
- $C_2 = [VM_1, Cloudlet_2]$
- $C_3 = [VM_2, Cloudlet_1]$
- $C_4 = [VM_2, Cloudlet_2]$

Le Tableau 5.3 montre les différentes combinaisons possibles avec les deux machines virtuelles (VM_1,VM_2), les deux *cloudlets* ($Cloudlet_1$,$Cloudlet_2$), les différents temps d'exécution et niveaux de charges CPU associés aux quatre combinaisons possibles. Dans cet exemple, si plusieurs couples de machines virtuelles et de *cloudlets*, tel que défini en C_3, sont lancés les uns à la suite des autres, avec par exemple un intervalle d'une seconde entre chaque, alors cela permet de faire augmenter la charge progressivement par palier de 2%. Une séquence différente d'exécution peut être par exemple générée en lançant cinq couples de type C_1, ce qui induira directement un pic de charge à 50% pendant 1.5 secondes.

C'est ainsi en combinant différents types de machines virtuelles et différents types de *cloudlets* que le scénario a pu être exactement reproduit dans le simulateur afin d'effectuer des comparaisons de comportement entre le simulateur et la machine physique *HOST* de façon rigoureuse.

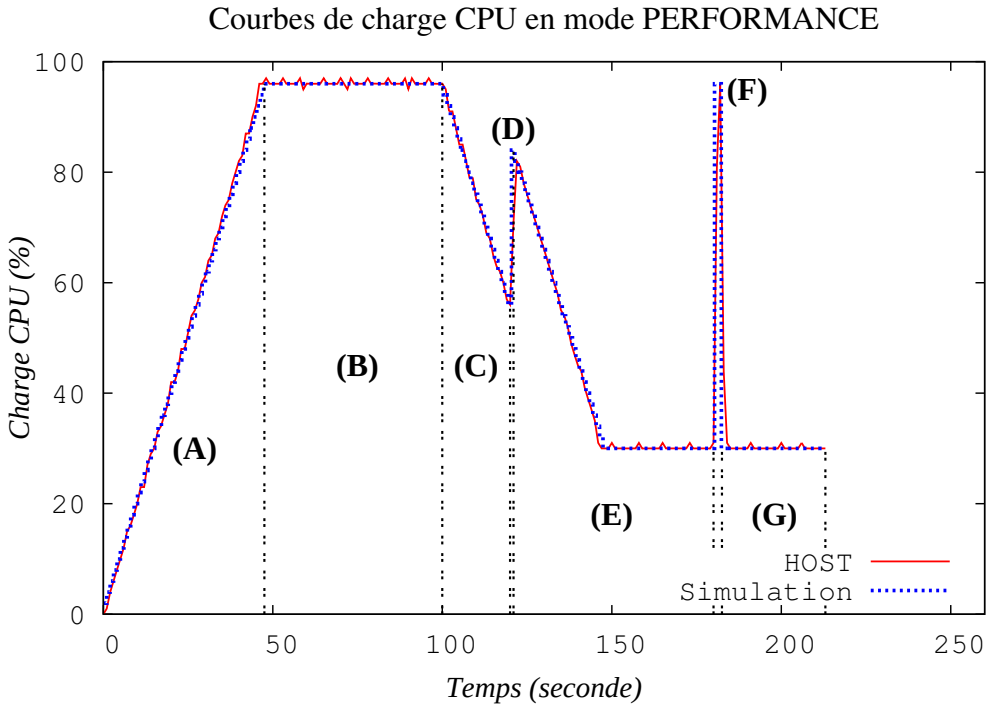


Figure 5.5 – Comparaison des courbes de charge CPU entre la machine physique *HOST* et la simulation dans CloudSim en utilisant le mode DVFS *Performance*.

Comparaison des courbes de charge CPU

La Figure 5.5 présente les courbes de charge CPU obtenues en utilisant la fréquence maximale (mode DVFS *Performance*) durant l'expérimentation sur la machine physique *HOST* et la simulation exécutée avec CloudSim. En utilisant le mode *Performance*, qui n'induit aucun changement dynamique de fréquence au cours du temps, le challenge était d'avoir exactement les mêmes courbes de charge CPU entre la simulation et l'expérimentation. En effet, à fréquence fixe et en prenant comme base la fréquence la plus élevée, les résultats de la simulation se devaient d'être très proches (voir égaux) des valeurs réelles d'expérimentation afin d'être certain de pouvoir poursuivre les autres simulations (en utilisant les autres modes DVFS) dans les meilleures conditions possibles. De plus, les différentes phases (de (A) à (G)) du scénario établi devaient être parfaitement distinguables.

Une fois ce résultat obtenu, la Figure 5.5 montrant uniquement d'infimes différences entre les deux courbes étant dûes à la fois à la différence d'intervalle entre deux mesures (1 seconde pour l'expérimentation contre 0.01s dans le simulateur) mais aussi aux petits défauts de précision du watt-mètre utilisé, les expérimentations et simulations utilisant les trois autres modes du DVFS ont pu être démarrées.

Sur la figure représentant les courbes de charges obtenues en mode *Conservative* (Figure 5.6), seules de petites variations de charge CPU peuvent être observées entre la phase (A) et (B) qui sont uniquement dûes à la différence de granularité des mesures de charge CPU entre la simulation et l'expérimentation et à l'augmentation progressive de la fréquence CPU appliquée par le gouverneur de ce mode. Par la suite,

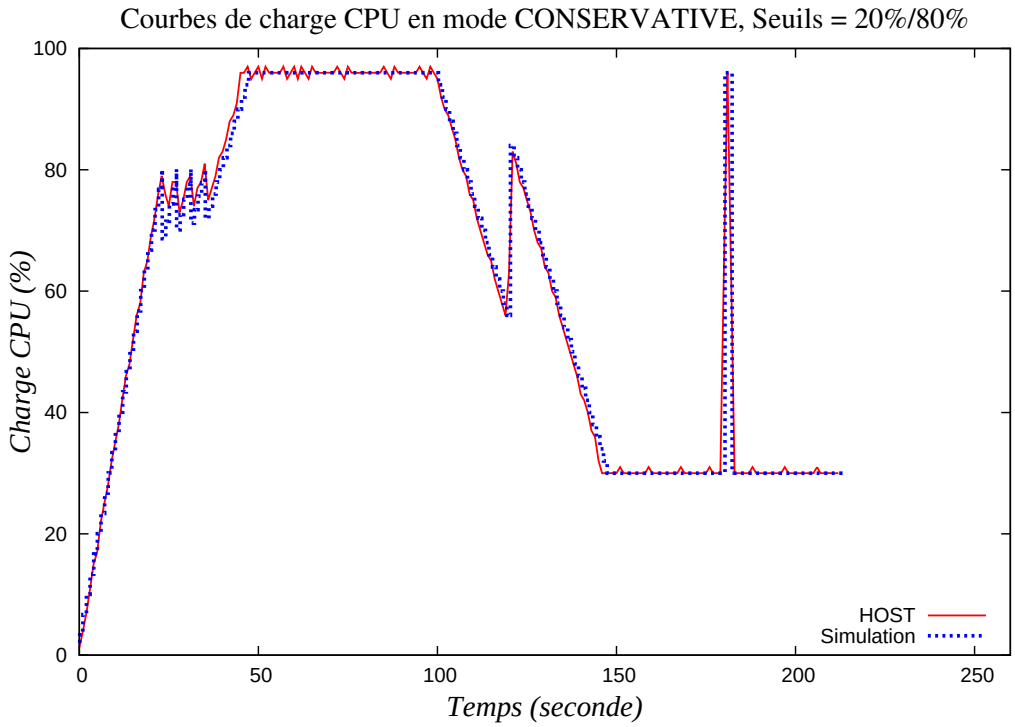


FIGURE 5.6 – Comparaison des courbes de charge CPU entre la machine physique *HOST* et la simulation dans CloudSim en utilisant le mode DVFS *Conservative*.

aucune différence ne peut être remarquée par rapport aux courbes obtenues avec le mode *Performance*. Cette similitude est justifiée par le fait que la charge CPU ne descend jamais en dessous du *down_threshold* (20% en mode *Conservative*), ce qui implique que la fréquence du CPU reste à son maximum (fréquence qu'il avait atteint à la fin de la phase (A)) jusqu'à la fin de la simulation.

Les résultats du mode *OnDemand* (Figure 5.7) montrent plus de variation de la charge CPU durant toute la durée du scénario. Le comportement intrinsèque du mode *OnDemand* favorise la performance des machines physiques en augmentant la fréquence CPU à son maximum, dès que la charge du CPU dépasse le seuil de décision pré-défini. Une fois que la charge CPU redescend en dessous de ce seuil, si celle-ci y reste au moins le nombre de fois défini par le paramètre *sampling down factor*, alors la fréquence est diminuée pas à pas (en passant par toutes les fréquences autorisées). Ce comportement spécifique à ce mode est facilement observable par exemple pendant la phase (C) du scénario, durant laquelle on peut remarquer de brusques variations de charge CPU en même temps que celle-ci diminue progressivement. Pendant cette phase de diminution progressive de la charge CPU, celle-ci passe en dessous des 95% (valeur du seuil de décision) ce qui provoque donc un abaissement de la fréquence. Chaque abaissement fait donc proportionnellement augmenter la charge CPU, et cela tant que la charge du CPU ne repasse pas au-dessus du seuil. Si un abaissement de fréquence fait repasser la charge CPU au delà de 95%, alors le *gouverneur* du mode *OnDemand* rétablit la fréquence CPU à sa valeur maximum. C'est exactement ce qui se passe durant cette phase, c'est pourquoi des pics de charge sont visibles (abaissement de fréquence), puis à nouveau une charge CPU plus faible lorsque que la fréquence maximum à été rétablie (signifiant que la

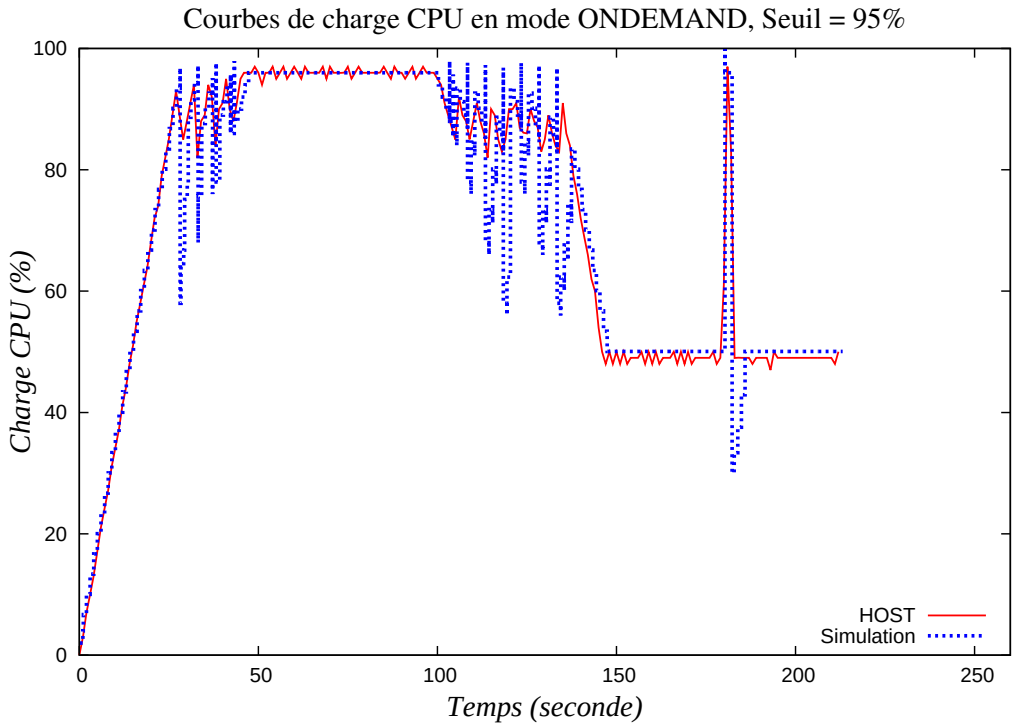


Figure 5.7 – Comparaison des courbes de charge CPU entre la machine physique *HOST* et la simulation dans CloudSim en utilisant le mode DVFS *OnDemand*.

charge CPU avait dépassé les 95% à $t-1$). Ces changements réguliers et interrompus de fréquence sont illustrés en Figure 5.9.

La dernière figure (Figure 5.8) compare les courbes de charge CPU utilisant le mode *PowerSave* du DVFS. Comme expliqué en Section 2.3.1 ce mode utilise la fréquence la plus basse admise par le CPU ce qui engendre donc très souvent un retard dans l'exécution des instructions dû à trop grand nombre d'instructions à traiter par rapport à la capacité de traitement que le CPU peut fournir. Toutes les phases du scénario sont exécutées séquentiellement, dès que du retard est provoqué dans l'une de ces phases, alors ce retard se répercute sur l'exécution des phases suivantes. En utilisant le mode *PowerSave* la charge CPU atteint très rapidement les 100% (à $t \simeq 30$s) alors que la charge maximum atteinte dans les autres modes est de 96% à $t \simeq 50$s. À partir de $t \simeq 30$s, toutes les instructions qui suivent sont donc exécutées en retard, faisant stagner la charge CPU à 100%. La phase (C) de diminution de charge est tout de même observable un court moment au environ de $t \simeq 65$s. Logiquement, le temps total d'exécution du scénario dans ce mode *PowerSave* est bien plus long que dans les autres modes utilisés.

Comparaison des résultats

Le tableau 5.4 regroupe les temps d'exécution et les consommations énergétiques relevées dans les différentes expérimentations et simulations. La dernière colonne du tableau établit une comparaison (sous forme de pourcentage d'erreur) des résultats des simulations en terme de calcul de consommation d'énergie par rapport aux valeurs mesurées pendant les expérimentations sur la machine physique *HOST*.

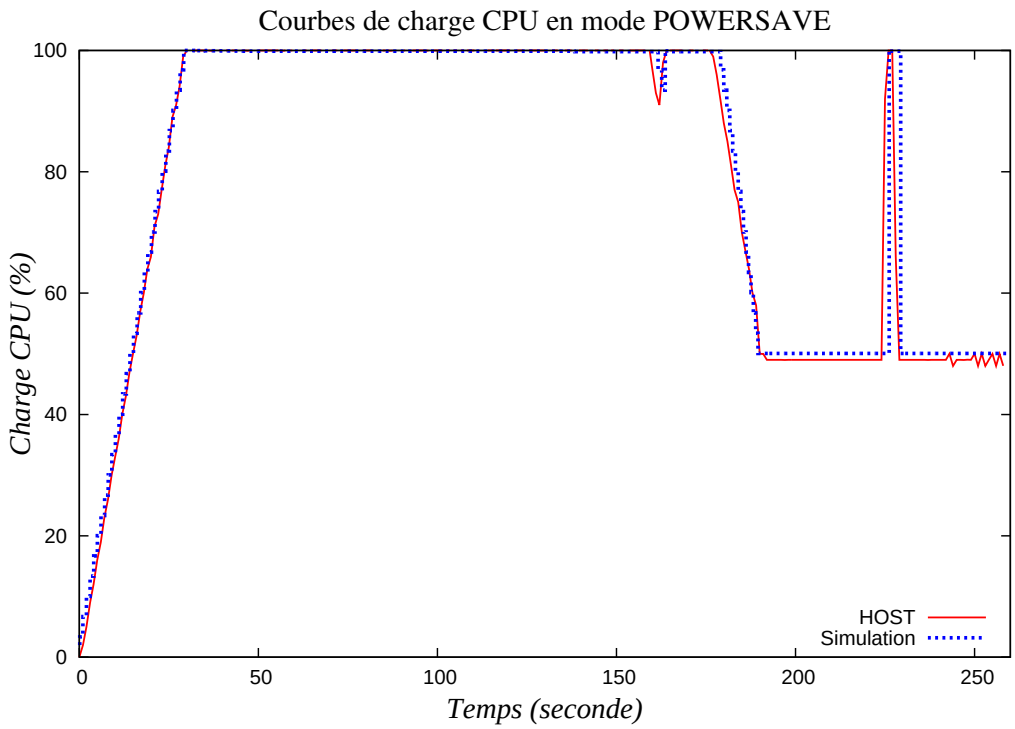


Figure 5.8 – Comparaison des courbes de charge CPU entre la machine physique *HOST* et la simulation dans CloudSim en utilisant le mode DVFS *PowerSave*.

Mode DVFS	*HOST*		**CloudSim**		**Erreur (%)**
	Durée (s)	Énergie (Wh)	Durée (s)	Énergie (Wh)	
Performance	213	5.72	213	5.61	1.92
OnDemand	213	5.57	213	5.49	1.43
Conservative	213	5.68	213	5.64	1.71
PowerSave	259	6.37	260	6.33	0.63

Table 5.4 – Comparaison des résultats entre la machine physique *HOST* et les simulations avec CloudSim dans chaque mode DVFS. Les pourcentages d'erreur donnés concernent les valeurs de consommation d'énergie des simulations par rapport à ceux mesurés lors des expérimentations.

Tout d'abord, il est intéressant de noté que les résultats proposés dans cette section suivent la logique des comportements des différents *gouverneurs* des modes du DVFS :

- Le mode *Performance* reproduit bien le scénario défini.
- Le mode *OnDemand* obtient le meilleur résultat en terme de consommation d'énergie tout en gardant le même temps d'exécution que le mode *Performance*.
- Le mode *Conservative* obtient un résultat de consommation énergétique à peine meilleur que le mode *Performance*.
- Le mode *PowerSave* a un temps d'exécution bien plus long que tous les autres modes et est logiquement aussi mauvais en terme de consommation d'énergie dans ce cas.

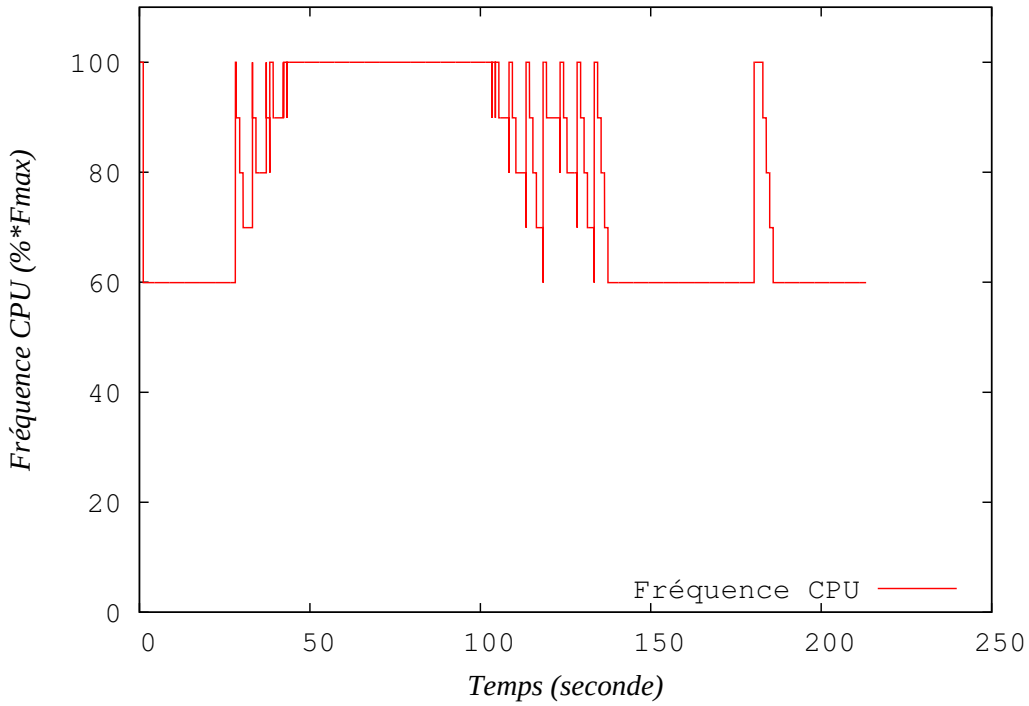


Figure 5.9 – Changements dynamiques de fréquence CPU durant la simulation avec CloudSim dans le mode de DVFS *OnDemand*.

Les modes *Conservative* (Figure 5.6) et *OnDemand* (Figure 5.7) obtiennent exactement le même résultat de temps d'exécution que le mode *Performance* pour deux raisons différentes. Comme expliqué précédemment, le mode *Conservative* fait augmenter et diminuer la fréquence CPU progressivement (pas à pas). Durant la phase (A) du scénario, la fréquence est donc progressivement augmentée jusqu'à atteindre sa valeur maximum possible. Par la suite, la charge ne descend jamais en dessous du *down_threshold* ce qui maintient la fréquence à sa valeur maximum (de la phase (B) à la phase (G)). Le résultat en terme de consommation d'énergie du mode *Conservative* est donc très proche de celui du mode *Performance*. Concernant le mode *OnDemand*, il empêche essentiellement une perte de performance en augmentant directement la fréquence du processeur à son maximum dès que possible, ce qui permet de ne pas dégrader le temps d'exécution. D'un point de vue consommation énergétique, le mode *OnDemand* obtient le meilleur résultat en raison de son comportement beaucoup plus fin pour diminuer la fréquence par rapport au mode *Conservative*. Les résultats du mode *PowerSave* peuvent à première vue paraître contradictoires, étant donné que théoriquement c'est le mode qui à un instant donné t permet à une machine physique de délivrer une puissance instantanée la plus faible possible, utilisant la fréquence CPU la plus basse. Expérimentalement, c'est justement cette très basse fréquence qui provoque un ralentissement des instructions à traiter et donc induit un temps d'exécution bien plus long. Ainsi, la consommation énergétique étant très dépendante du temps d'utilisation, le mode *PowerSave* est celui qui consomme le plus d'énergie dû à son temps d'exécution très long.

Outre ces constatations et explications de ces résultats, il est également très important de remarquer qu'il existe une réelle cohérence entre les résultats des expérimentations et ceux obtenus par simulations. Les résultats affichés dans le tableau 5.4 permettent de déduire que cette phase de validation du DVFS dans CloudSim a été conduite dans de bonnes conditions, permettant des mesures de puissances réelles précises et de modéliser précisément les simulations qui amènent à un taux d'erreur de calcul de consommation d'énergie de 1.92% dans le pire des cas.

5.3.2 Graphe de tâches

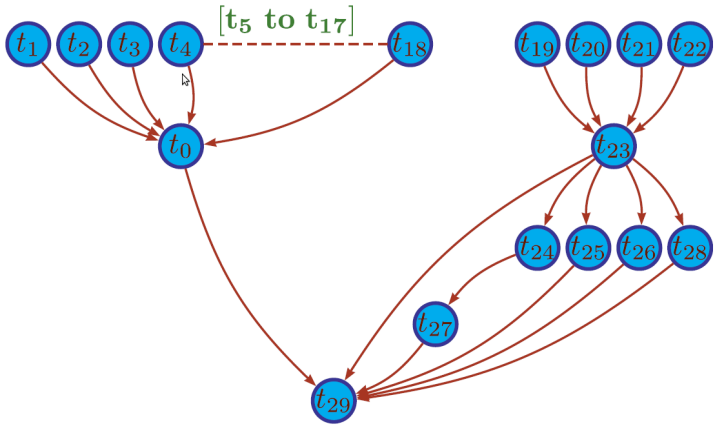


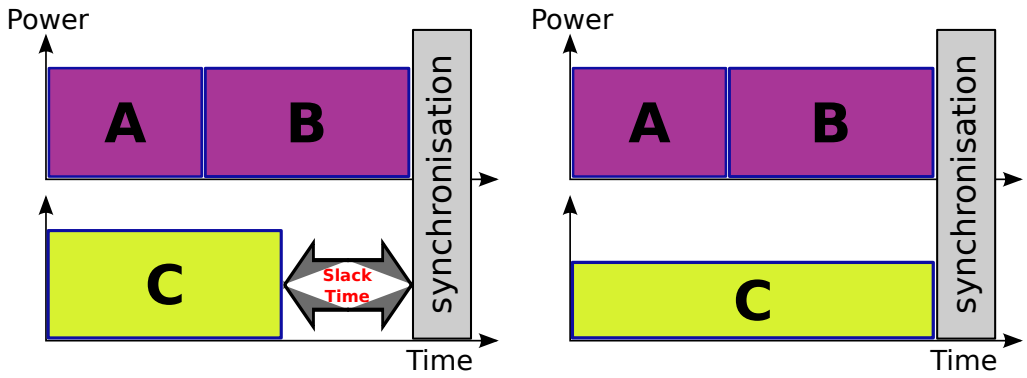
Figure 5.10 – Graphe du DAG *Sipht30* [2], composé de 30 tâches, utilisé pour les simulations.

Cette section présente une utilisation du DVFS appliquée à un DAG (*Direct Acyclic Graph*), uniquement par simulation. Le but des simulations d'exécution de DAG est de démontrer l'efficacité du DVFS, dans différentes configurations, afin d'optimiser l'exécution du DAG et ainsi de réduire la consommation d'énergie totale. De plus, une autre utilité de ce chapitre est de montrer que les résultats ne sont pas une finalité en soi, mais que l'optimisation de l'exécution du DAG obtenue par simulation est utile et pourrait tout à fait servir de base pour une exécution sur une réelle plate-forme.

Une application modélisée par un DAG est une suite de tâches ordonnées représentées par un graphe composé de nœuds (Figure 5.10). Une tâche peut avoir un ou plusieurs *fils* qui doivent être exécutés après leur tâche parente. Ces dépendances entres les tâches sont transmises par fichiers. Ces derniers sont générés à la fin de l'exécution d'une tâche parente et utilisés comme données d'entrée pour l'exécution des *fils*. La fin d'exécution d'un DAG se termine toujours par une tâche qui ne possède pas de *fils*, appelée *feuille*.

Le plus long chemin, établi depuis une tâche source (qui ne possède pas de parents) jusqu'à une *feuille* est appelé *chemin critique* (Figure 5.12). Le temps d'exécution minimum total du DAG est donc borné par la somme des temps d'exécution de chaque tâche, appelée *tâche critique* composant ce *chemin critique*. Si un DAG possède plusieurs *chemins critiques*, un *sous-graph critique* contenant tous les *chemins critiques* est défini. Considérant que ces chemins sont invariants, le temps total d'exécution du DAG est égal à l'instant t de terminaison de la dernière tâche de ces *chemins critiques*. Chaque tâche en dehors de ce ou ces chemins est appelée *tâche non critique*.

2. https ://confluence.pegasus.isi.edu/display/pegasus/SIPHT+Characterization



(a) Illustration du *Slack-Time* d'une tâche *non critique*.

(b) Illustration du gain énergétique à appliquer le DVFS sur une tâche *non critique* en abaissant la fréquence CPU de la machine physique sur laquelle elle est exécutée de façon à augmenter son temps d'exécution de la valeur du *Slack-Time*.

FIGURE 5.11 – Illustrations d'exécutions avec et sans DVFS d'une tâche *non-critique*

Ces *tâches non critiques* (Figure 5.11a) n'influencent donc pas le temps nécessaire à l'exécution du DAG et permettent d'introduire la notion de *Slack-Time*. Le *Slack-Time* représente le temps écoulé entre la fin de la tâche *non-critique* concernée et le début d'un de ses *fils* appartenant à un *chemin critique*. Partant de ce fait, l'utilisation du DVFS va permettre de ralentir l'exécution de ces dernières (Figure 5.11b) afin de réduire la consommation d'énergie durant leur exécution et ainsi diminuer la consommation d'énergie globale par rapport à une exécution sans DVFS. Ce procédé doit bien sûr être établi de façon à ne pas augmenter le temps total d'exécution du DAG.

Le modèle de *workflow* considère les dépendances entre les tâches, le temps d'exécution de chaque tâche, et la quantité de données que chaque tâche doit envoyer à ses *fils*. Le temps requis pour transférer les données depuis la machine virtuelle de la tâche *parent* à la machine virtuelle de la tâche *fils* défini le temps de communication entre les tâches. Ce temps est égal à 0 si deux tâches sont exécutées sur la même machine virtuelle, sinon ce temps de communication est calculé et pris en compte lors de l'analyse du DAG pour déterminer le ou les *chemins critiques*.

Afin d'établir une borne inférieure théorique sur l'économie d'énergie rendue possible par l'utilisation du DVFS lors de l'exécution du DAG, l'algorithme de placement utilisé alloue chaque tâche à une machine virtuelle différente, et chaque machine virtuelle est placée sur une machine physique différente. Cela force à transférer les données à la fin de chaque exécution de tâche.

L'application utilisée sous forme de DAG pour ces simulation est appelée *Sipht30* et est illustrée par le graph de la Figure 5.10. Ce DAG est une instance des applications *Sipht* qui sont utilisés pour automatiser l'encodage de gènes. Ce *workflow* utilise un assez grand nombre d'applications, mais a un comportement très stable à la fois en terme de temps exécution et de tailles données échangées[3]. Les temps de transfert de données sont inférieurs aux temps d'exécution des tâches.

3. https://confluence.pegasus.isi.edu/display/pegasus/SIPHT+Characterization.

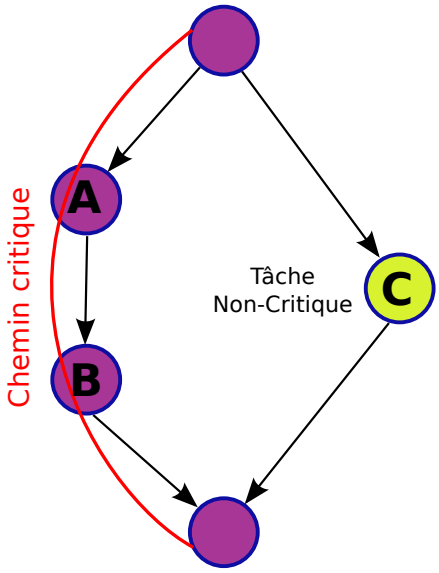


Figure 5.12 – Illustration d'un chemin critique.

***Slack-Time* : Algorithme et fréquence optimale**

Le ralentissement des tâches *non-critiques* est basé sur le principe de trouver la fréquence CPU permettant de consommer le moins d'énergie possible. Dans ce cas d'étude cette fréquence est appelée fréquence optimale et dépend de plusieurs paramètres :

- Les caractéristiques de puissance des machines physiques utilisées
- La topologie du *workflow*
- Le ratio entre la durée de la tâche et la durée du *slack-time*

Pour cette expérimentation permettant de connaître la fréquence optimale, les caractéristiques de puissance des machines physiques du site Grid'5000 de Reims ont été utilisées (Tableau 5.6). La méthodologie adoptée a donc été de fixer un temps d'exécution de tâche (T_{task}), de faire varier la durée du *Slack-Time*, puis de calculer l'énergie qui a été nécessaire à l'exécution de cette tâche. Cela est effectué pour chacune des fréquences disponibles sur le CPU des machines physiques de Reims. La figure obtenue (Figure 5.13) montre donc les différentes fréquences optimales (celles qui ont donné les consommations d'énergie les plus faibles) en fonction du ratio $\frac{Slack-Time}{T_{task}}$. Étant donné le nombre fini de fréquences disponibles, la figure affiche une utilisation de fréquences optimales discrètes.

La méthode utilisée pour calculer le *Slack-Time* de chaque tâche *non-critique* a été inspirée par celle de Kimura *et al.* [73], qui présentent une étude sur les économies d'énergie employant le DVFS et les *Slack-Time* dans les DAGs. D'autres études sur les *workflow overhead* ont été menées par by Chen et Deelman [29], puis Nerieri et al. [100].

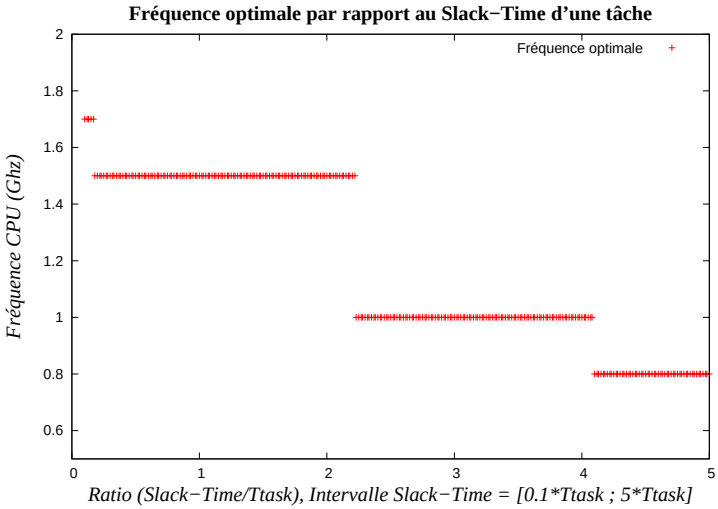


Figure 5.13 – Graphique des fréquences optimales obtenues en utilisant les valeurs de puissance et de fréquences du site Grid'5000 de Reims en fonction du *Slack-Time*. Les valeurs de *Slack-Time* étudiées sont comprises dans l'intervalle $[(0.1 * T_{task}); (5 * T_{task})]$. T_{task} est le temps d'exécution fixé pour une tâche et l'unité de l'axe X est le ratio $\frac{Slack-Time}{T_{task}}$.

Résultats et comparaisons

Le Tableau 5.5 affiche les résultats de consommation d'énergie obtenus en utilisant les différents mode de fonctionnement du DVFS appliqués à l'exécution du DAG *Sipht30*. Les simulations de DAG on été exécutées en utilisant trois mode de DVFS différents. Premièrement en mode *Performance* qui exécute donc toutes les tâches en utilisant la fréquence CPU maximum des machines physiques concernées. Le mode dynamique *OnDemand* a également été évalué pour le comparer avec le mode *UserSpace*, qui permet de fixer la fréquence des machines physique à une valeur choisie. Ce mode à donc été utilisé en relation avec le calcul de la valeur du *Slack-Time* de chaque tâche *non-critique*. En effet, comme illustré sur les Figures 5.11a et 5.11b, une fréquence plus basse (F_{opt}), ralentissant l'exécution de la tâche concernée peut être choisie en fonction de la durée de son *Slack-Time*. Cela permet donc de déterminer pour chaque tâche *non-critique* la fréquence optimale F_{opt} à utiliser.

DAG *Sipht_30*	**DVFS Modes**		
	Performance	*UserSpace* (F_{opt})	*OnDemand*
Energy (Wh)	3241	2817	2751
Gain (%)	⊘	13.1%	15.1%

Table 5.5 – Comparaison des résultats de consommation d'énergie (en Wh) obtenus en utilisant 3 modes DVFS différents. Les valeurs de la ligne "gains" sont calculées par rapport à la valeur de consommation d'énergie du mode *Performance*.

.

Le mode *Performance* donne une consommation énergétique de 3241 Wh. En utilisant ce mode, la fréquence CPU utilisée est la plus haute et est fixe durant toute l'exécution du DAG. Cela amène

inévitablement à une consommation d'énergie assez élevée, car aucun ajustement de fréquence n'est appliqué.

Le première constatation est de voir que le mode *OnDemand* est celui qui donne le meilleur résultat en terme de consommation d'énergie. En effet, le mode *OnDemand* permet d'attribuer la fréquence maximum durant les phases calculs intensifs, puis de faire fonctionner le CPU à la plus faible fréquence lorsqu'il a fini d'exécuter la tâche qu'il lui était affectée. Cela permet donc d'avoir une capacité de calcul importante pour exécuter les tâches puis d'avoir la consommation énergétique la plus faible possible lorsque que les tâches sont finies. Le mode *OnDemand* obtient un gain de 15.1% par rapport au mode *Performance*.

L'utilisation du mode *UserSpace* permet un gain de 13.1% par rapport au mode *Performance*, mais obtient un résultat de consommation énergétique un peu moins bon que le mode *OnDemand*, bien qu'il utilise la fréquence optimale F_{opt} calculée. En effet, en utilisant le mode *UserSpace* la fréquence optimale peut être calculée et utilisée pour ralentir la tâche concernée et donc optimiser au maximum la consommation énergétique durant l'exécution de cette tâche. Seulement, une fois l'exécution de la tâche terminée, le CPU continue de fonctionner à cette fréquence bien qu'il n'ait plus aucune tâche à traiter. C'est donc les phases pendant lesquelles le CPU est inutilisé que le mode *UserSpace* redevient moins efficace énergiquement que le mode *OnDemand*.

Au vu de ces résultats, on peut constater que l'utilisation du DVFS peut s'avérer être très utile pour l'exécution de DAG. De plus, l'exécution du DAG, utilisant le calcul de la fréquence optimale d'exécution des tâches *non-critiques* associée à leur valeur de *Slack-Time*, est très performante étant donné qu'elle obtient un résultat de consommation énergétique presque aussi bon que le mode *OnDemand*. Enfin, il est intéressant de constater qu'une méthode alliant le calcul de la fréquence optimale et un abaissement de la fréquence CPU au minimum une fois les tâches terminées pourrait s'avérer encore plus performante. Comme introduit en début de section, ces résultats et ces analyses ne s'arrêtent pas à l'unique but de démontrer l'utilité du DVFS pour l'exécution de DAG, mais que ces différentes remarques et constatations quant à la configuration du DAG et du DVFS obtenues grâce à ces simulations peuvent servir de base pour l'optimisation d'exécution de DAG sur une réelle plate-forme.

5.3.3 Application parallèle d'électromagnétisme réelle : Expérimentations sur Grid'5000 et simulations

Cette section présente les expérimentations menées sur la plate-forme Grid'5000 utilisant une application parallèle distribuée d'électromagnétisme, *Transmission Line Matrix* (TLM), spécifiquement implémentée pour ce type d'architecture, et les simulations associées à celles-ci. L'objectif de cette section est de présenter un troisième cas d'utilisation plus complexe du DVFS, en conditions réelles par l'utilisation de la grille de calcul Grid'5000, et par simulation en modélisant le comportement de l'application distribuée. Un modèle analytique dédié à la TLM est également utilisé.

Cette section propose donc une présentation de la TLM, une présentation des caractéristiques du site Grid'5000 de Reims sur lequel celle-ci a été exécutée, un modèle analytique permettant d'approximer le comportement de cette application parallèle, puis une modélisation de la TLM au sein du simulateur CloudSim.

Les résultats obtenus sont comparés en terme de temps d'exécution et de consommation d'énergie utilisant différents modes de DVFS.

TLM description

La méthode *Transmission Line Matrix* est une méthode numérique pour la simulation électromagnétique qui représente un environnement de propagation de champs électromagnétiques à travers un réseau de lignes de transmission. Ce modèle de propagation réside sur l'équivalence qui existe entre les champs électriques et magnétiques ainsi qu'entre les tensions et les courants dans un réseau de lignes de transmission. Une présentation détaillée de la méthode TLM, est décrite dans [63]. La résolution de cette méthode de manière parallèle sur une grille de calcul, est basée sur un découpage de la structure suivant les trois axes. Les volumes obtenus sont assimilés à des tâches, exécutées chacune sur une machine, un CPU, ou un cœur de processeur, échangeant des informations via la bibliothèque MPI (*Message Passing Interface*).

Caractéristiques applicative et matérielle

La bibliothèque MPI utilisée pour l'échange d'informations entre les nœuds de la TLM est *OpenMPI*. Un point critique de cette bibliothèque qui doit être considéré est le *Pooling Time*. Le *Pooling Time* est le temps pendant lequel une tâche est en attente de réception de données envoyées par ses voisins. Bien que cela puisse être considéré comme un temps de sommeil, cela n'est pas le cas lorsque la bibliothèque *OpenMPI* bibliothèque est utilisée. Au contraire, durant ce temps le CPU est utilisé à 100% comme s'il était en phase de calcul intensif. C'est pourquoi il est très important de tenir compte de ce paramètre lors des expérimentations énergétiques. En effet, un abaissement de fréquence durant le *Pooling Time* est beaucoup moins efficace que dans une phase où la charge du CPU est à 0% d'utilisation. Cette caractéristique de la bibliothèque MPI et une implémentation d'une version *non-pooling* sont abordées par White et Bova dans [144].

Pour la partie expérimentation, les machines physiques du site Grid'5000 [60, 25] de Reims ont été utilisées. Les caractéristiques de puissance de ces machines sont décrites dans le tableau 5.6. Une autre caractéristique de ces machines concernant le DVFS se doit d'être mentionnée : en effet, comme expliqué précédemment, sur certaine architecture le DVFS n'affecte pas seulement la fréquence du CPU mais également celle du FSB entraînant un ralentissement d'autres composants. Après de nombreuses campagnes d'expérimentations sur ces machines, les différents tests pour comprendre le comportement du DVFS sur ces machines ont démontré que lorsqu'on modifiait la fréquence du CPU, la vitesse des E/S était aussi affectée par ce changement de fréquence. Ce comportement évoqué précédemment à donc été mis en évidence sur ces machines dont les détails de l'architecture sont accessibles sur le site de Grid'5000[4].

Une fois toutes ces caractéristiques connues, deux modèles peuvent être définis pour étudier la consommation d'énergie. Le premier est le modèle analytique qui permet un calcul rapide de la consommation d'énergie en se basant sur les équations du modèle de prédiction. Le second est un modèle évènementiel, mis en place par la modélisation de la TLM dans CloudSim. Le comportement de la TLM est alors modélisé le plus précisément possible afin de reproduire le comportement de l'application dans le simulateur. Cela

4. https://www.grid5000.fr/mediawiki/index.php/Reims:Hardware

Reims Grid'5000						
Fréquences disponibles (GHz)		0.8	1.0	1.2	1.5	1.7
Puissance (W)	**Pidle**	140	146	153	159	167
	Pfull	228	238	249	260	272

TABLE 5.6 – Fréquences CPU disponibles sur le site Grid'5000 de Reims et les puissances associées à 0% (**Pidle**) et 100% (**Pfull**) d'utilisation des 24 cœurs d'un CPU.

se fait par un enchaînement de séquences d'évènements. Les résultats de ces deux modèles sont analysés et comparés aux valeurs réelles des expérimentations effectuées sur la grille de calcul en section 5.3.3.

Modèle analytique

Le modèle introduit ci-dessous permet le calcul de la consommation d'énergie par la TLM dans une configuration donnée, si la fréquence utilisée est connue. Comme expliqué dans la section précédente, le modèle doit prendre en compte le *Pooling Time* et la relation entre les changements de fréquence CPU et le débit des entrées/sorties.

Description et notations du modèle :

- H : Nœud utilisé
- T_{cpu} and T_{net} : Temps CPU et Réseau
- D_{disk} : Quantité de données à écrire sur le disque (en nombre d'éléments)
- $Th_{disk}(f_i)$: Débit du disque dur à la fréquence f_i.
- $F = \{f_1, f_2, ..., f_{n-1}, f_n\}$: Fréquences CPU utilisables sur le nœud H, avec $f_{n-1} < f_n$.
- $l^H_{cpu}(f_i)$ and $h^H_{cpu}(f_i)$: Puissance délivrée par le nœud H à 0% et 100% d'utilisation CPU.
- $l^H_{disk}(f_i)$: Puissance délivrée par la nœud H quand le disque dur est utilisé.

Le modèle de prédiction d'utilisation CPU, réseau et disque de la TLM utilisé est celui présentée par Alexandru dans [3] :

- $T_{cpu} = C_1 + n_l n_x n_y n_z C_2$
- $T_{net} = \left(L + \frac{n_x n_y}{D}\right) * 4n_l$
- $D_{disk} = n_x * n_y * n_z$

avec,

- n_x, n_y, n_z les dimension de l'environnement TLM
- n_l le nombre d'itérations
- C_1 et C_2 des constantes estimées à partir d'expérimentations précédentes
- L la Latence réseau
- D le débit du réseau.

L'équation de la consommation d'énergie E est donc formulée comme suit :

$$E = T_{net} * h^H_{cpu}(f_i) + (T_{cpu} * f_n) \left[\frac{h^H_{cpu}(f_i)}{f_i} \right] + \left[\frac{D_{disk}}{Th_{disk}(f_i)} \right] * l^H_{disk}(f_i) \quad (5.3.1)$$

Ce modèle a dont été utilisé en utilisant une taille de problème donné présentée dans la section suivante. En utilisant les données d'entrée de la TLM et les données de configuration, une estimation de la consommation d'énergie générée par l'exécution de la TLM peut être estimée.

La section suivante présente la configuration de la TLM telle qu'elle a été exécutée réellement sur la grille de calcul Grid'5000.

Expérimentations sur Grid'5000

Les expérimentations ont été menées sur le site Grid'5000 de Reims, dans la configuration est la suivante : *HP ProLiant DL165 G7, CPU : AMD Opteron 6164 HE 1.7GHz, 12MB L3 cache, 44 nœuds avec 2 CPUs de 12 cœurs per CPU (1056 cœurs).*

La TLM a été exécutée sur 2 nœuds, utilisés tous les deux à 100% de leur capacité (soit 48 cœurs) : une tâche TLM utilisant chacune 1 cœur.

Les valeurs des paramètres d'entrées utilisés sont les suivantes :

- n_x=172
- n_y=90
- n_z=12
- n_l=26000
- D=700 Mbit/s
- L=4*10^{-5} seconde

Le déploiement de la TLM sur Grid'5000 a été fait dans un environnement *Kadeploy* [5], en utilisant deux modes de DVFS différents :

– Mode *Performance* : Fréquence maximum de 1.7GHz
– Mode *OnDemand*

Durant l'exécution de ces expérimentations, la puissance délivrée par les nœuds est récupérée à intervalles réguliers et sauvegarder afin de pouvoir calculer la consommation d'énergie totale à la fin de l'expérimentation. En effet, les nœuds du site Grid'5000 de Reims sont équipés de PDUs (*Power Distribution Units*) qui permettent de connaître la puissance qu'ils délivrent à un instant t. Ces PDUs ont une précision d'environ 6 Watts, et actualisent leurs valeur toutes les 3 secondes.

Modèle événementiel

Le modèle évènementiel représente la façon dont l'application TLM a été modélisée dans le simulateur CloudSim (Figure 5.14). Lors des simulations, les 24 tâches sont exécutées par une machine physique avec

5. Kadeploy, un système de déploiement dédié aux *cluster* et aux grilles de calcul : https ://gforge.inria.fr/projects/kadeploy/

24 cœurs. Sur un nœud de la grille, la quantité de données E/S est écrite sur un disque partagé par les 24 tâches d'un nœud. Par conséquent, dans le simulateur cette caractéristique a été représentée par une seule machine physique partagée pour toutes les tâches. Cette méthode a été appliquée car de la gestion du stockage dans le simulateur ne permet pas de ralentir la vitesse des entrées/sorties lors de l'utilisation DVFS. Comme expliqué dans la section 5.3.3, le débit du disque est ralenti proportionnellement à la fréquence CPU utilisée. Pour reproduire ce comportement, le DVFS doit donc être activé sur l'hôte qui simule la capacité du disque afin de ralentir les E/S proportionnellement. Enfin, le trafic réseau généré par toutes les communications MPI entre les tâches et les nœuds est représenté par une seule machine physique dans CloudSim partagée par toutes les autres. Grâce à cette architecture de simulation, toutes les capacités de CPU, disques et réseau sont respectées.

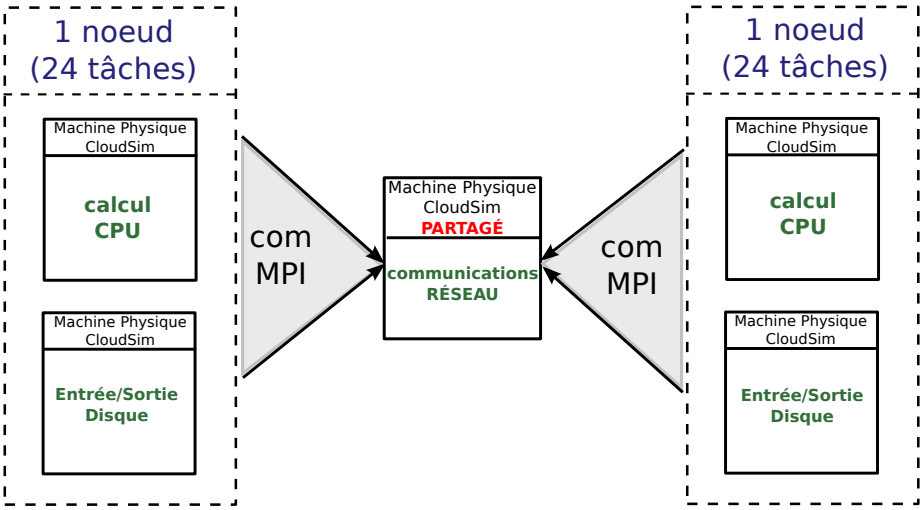


Figure 5.14 – Représentation graphique de la décomposition du fonctionnement de la TLM telle que modélisée dans le simulateur.

La section suivante présente les résultats obtenus et établit une comparaison entre les différentes valeurs de temps d'exécution et de consommation d'énergie obtenus avec l'utilisation des modes *Performance* et *OnDemand* du DVFS.

Résultats

Le tableau 5.7 affiche les résultats obtenus pour les deux modèles définis, et ceux des exécutions de la TLM sur Grid'5000. Le modèle analytique permet d'obtenir très rapidement une estimation du temps d'exécution et de l'énergie consommée en utilisant les équations liées à la taille de la structure TLM et les caractéristiques de puissance des composants physiques utilisés (processeur, réseau, et disque). Le modèle évènementiel est utilisé dans le simulateur CloudSim avec une représentation virtuelle de l'application et du comportement du matériel comme expliqué dans la section 5.3.3. Ce modèle permet la simulation des deux modes de DVFS, *Performance* et *OnDemand*, également utilisés lors des expérimentations réelles, et donne à la fois des résultats précis de temps d'exécution et de consommation d'énergie. Le modèle analytique donne également de bons résultats pour le mode *performance*, mais à la différence du modèle évènementiel,

il peut être utilisé uniquement dans ce mode. En effet, les changements de fréquence dynamiques relatives à la charge du processeur, générés lors de l'utilisation du mode *OnDemand*, ne peuvent pas être modélisés avec précision avec ce modèle.

Concernant les valeurs de la consommation d'énergie, le mode *OnDemand* donne des résultats moins bons que le mode *Performance*. Comme expliqué dans la Section 5.3.3, cela est causé par le ralentissement des E/S lorsque la fréquence CPU diminue et par le *Pooling Time* pendant les communications MPI. Ces résultats méritent donc plus d'explications. La plupart du temps, le mode *OnDemand* permet toujours d'obtenir de meilleurs résultats de consommation d'énergie. Sur le site de Reims, au cours d'une phase de charge CPU intensive, le mode *OnDemand* fonctionne comme prévu, diminuant et augmentant dynamiquement la fréquence relativement à son seuil de décision. Cependant, quand une tâche TLM attend des données de ses voisins, la charge du processeur reste à 100% en raison des caractéristiques du *Pooling*. Dans cette situation, le gouverneur du mode *OnDemand* ne change pas la fréquence du CPU et introduit donc une forte consommation d'énergie. Au cours d'une phase E/S, la charge CPU retombe naturellement à 0% et le gouverneur du mode *OnDemand* diminue rapidement la fréquence du CPU, ce qui conduit à ralentir les E/S proportionnellement à la nouvelle fréquence. Ce comportement qui apparaît pendant les phases de communications réseau et de E/S explique l'inefficacité et les mauvais résultats du mode *OnDemand*.

	Modes DVFS							
	Performance				***OnDemand***			
	Temps (s)		Énergie (Wh)		Temps (s)		Énergie (Wh)	
	Valeur	Erreur	Valeur	Erreur	Valeur	Erreur	Valeur	Erreur
Simulation	3790	-0.07%	478	-1.44%	4932	-0.1%	613	-0.8%
Analytique	3882	2.69%	499	2.88%	⊘	⊘	⊘	⊘
Expérimentation	3793	⊘	485	⊘	4937	⊘	618	⊘

Table 5.7 – Comparaison entre les résultats obtenus lors des expérimentations Grid'5000 sur le site de Reims, les résultats du modèle analytique et les simulations avec Cloudsim de la TLM. Les pourcentages d'erreur sont calculés par rapport aux valeurs des résultats des expérimentations sur Grid'5000. Les résultats expérimentaux sont basés sur une campagne de 20 exécutions de la TLM sur la grille, donnant respectivement des valeurs d'écart-type de 150s et 22Wh pour le temps d'exécution et la consommation énergétique dans chacun des modes DVFS utilisés.

5.4 Bilan

L'enjeu de ce chapitre était de présenté le simulateur CloudSim, largement utilisé au cours de cette thèse, d'introduire ses caractéristiques mais surtout d'expliquer les nouveaux outils qui ont dû être intégrés afin qu'il regroupe l'ensemble des fonctionnalités voulues. L'aboutissement de l'intégration de ces nouvelles fonctionnalités font de CloudSim un simulateur de *Cloud* complet, regroupant de nombreux outils de gestion de consommation énergétique non-implémentés dans les autres simulateurs à l'heure actuelle. En effet, l'intégration du DVFS et les reconfigurations de machines virtuelles au sein de ce simulateur permettent de reproduire à l'identique le comportement de changement dynamique de fréquence tel qu'il est implémenté dans le noyau Linux tout en adaptant la capacité des machines virtuelles lorsque cela est nécessaire. En terme de simulations énergétiques, ces fonctionnalités ouvrent de nombreuses possibilités

d'étude d'une granularité très fine au niveau des changements de fréquence CPU et de la consommation énergétique des machines physiques. Le principe de reconfiguration de machines virtuelles en terme de capacité CPU est fortement relié à l'implémentation et aux comportements intrinsèques des différents gouverneurs du DVFS dûs aux variations de capacité de traitement des machines physiques que ceux-ci provoquent. Tout cela représente une avancée certaine dans le domaine des études menées par simulations, s'attaquant aux problèmes de réduction de consommation d'énergie au sein d'un environnement *Cloud Computing*.

Les contributions aux caractéristiques de CloudSim ont permis d'établir une collaboration avec le laboratoire *CLOUDS Laboratory* de Melbourne en Australie (dirigé par Prof. Rajkumar Buyya), et de publier l'ensemble de ces travaux dans une revue internationale : *Energy-aware simulation with DVFS*. Simulation Modelling Practice and Theory, Volume 39, pages 76-91, December 2013. De plus, le code source intégrant les fonctionnalités utilisées dans ce chapitre est disponible sur le site internet de CloudSim [6]

Le chapitre suivant expose les simulations effectuées afin de pouvoir évaluer l'évolution des valeurs des paramètres de QoS pris en compte, en fonction des différents algorithmes de placement utilisés pour l'allocation et la ré-allocation des machines virtuelles sur les machines physiques.

6. Code source de CloudSim_DVFS : http ://www.cloudbus.org/cloudsim/CloudSim_DVFS.rar

6

Simulations d'ordonnancement Cloud sous contraintes de QoS

Sommaire

Ce chapitre détaille les simulations d'ordonnancement de machines virtuelles sous contraintes de qualité de service. Les quatre paramètres de qualité de service évalués lors de ces simulations sont ceux présentés en Section 4.2. Les différents résultats qui suivent ont été obtenus en déroulant des simulations utilisant les trois algorithmes de placement présentés en Chapitre 4 : l'algorithme génétique, Round-Robin (RR) et Best-Fit Sorted (BFS). Le simulateur CloudSim a été utilisé, incluant les améliorations qu'il lui ont été apportées, afin de pouvoir analyser l'évolution de ces paramètres de QoS au cours du temps. En effet, l'utilisation de ce simulateur (contrairement aux résultats présentés en Chapitre 4), permet de dérouler l'exécution des machines virtuelles en fonction de leur allocation tout en calculant les valeurs des métriques à chaque pas de simulation. Cette phase d'évaluation par simulation met donc en jeu l'ensemble des travaux présentés dans les chapitres précédents, alliant l'utilisation de CloudSim et son implémentation du DVFS, les reconfigurations de machines virtuelles, les algorithmes de placements et les métriques de qualité de service. L'évaluation par simulation de paramètres de qualité de service amène également la possibilité d'analyser la pertinence et l'influence de chaque métrique les unes par rapport aux autres, mais également d'apprécier l'impact du comportement de chaque algorithme de placement sur l'évolution de celles-ci ainsi que sur le déroulement des exécutions des machines virtuelles.

Ce chapitre est découpé en trois sections. La première présente la méthodologie appliquée pour exécuter les simulations, la seconde expose et commente les graphiques de résultats de ces simulations utilisant à

la fois les algorithmes gloutons et différentes configurations de l'algorithme génétique, en exposant des analyses détaillées de l'impact des métriques, des algorithmes d'ordonnancement utilisés et des différentes versions monos et multi-objectifs du GA. Enfin, la dernière section propose un jugement global sur l'approche multi-objectifs proposée dans ce chapitre afin de synthétiser l'ensemble des analyses exposées.

6.1 Méthodologie

Cette section de méthodologie expose toutes les étapes de configurations nécessaires de manière à définir l'environnement de simulation. Une fois ces configurations figées, celles-ci sont utilisées pour toutes les simulations afin d'être dans de bonnes conditions de comparaison.

Les sections suivantes détaillent :

- La configuration des machines physiques
- La configuration des machines virtuelles
- La configuration des paramètres globaux de l'algorithme génétique
- L'utilisation des algorithmes d'ordonnancement et du DVFS dans le simulateur

6.1.1 Configuration des machines physiques

Reprenant les caractéristiques décrites dans le Chapitre 3, les propriétés des machines physiques sont présentées ci-dessous :

- Nombre de machines physiques : 110
- Capacités maximums de CPU (en MIPS) et de Mémoire (en Mo) : 2000 MIPS / 2500Mo
- Gamme des fréquences CPU, rappelées en Tableau 6.2
- Hétérogénéité des puissances délivrées par les machines physiques en fonction des fréquences CPU, résumées en Tableau 6.1

Hétérogénéité des machines physiques

Concernant les valeurs de puissances délivrées par les machines physiques, les caractéristiques utilisées sont celles des machines du site Grid'5000 de Reims, rappelées dans le Tableau 6.2. Une hétérogénéité des caractéristiques de puissance des machines physiques a été introduite afin de se rapprocher de la composition des *data centers* actuels pouvant regrouper des machines de différentes générations. Ainsi, cinq types différents, délivrant plus ou moins de Watts que le modèle de base : le type 0 délivre 20% de moins, le type 5 délivre 20% de plus que le type 2 correspondant exactement aux caractéristiques de base des machines de Grid'5000. Ces cinq types sont détaillés dans le Tableau 6.1. Chacune des 110 machines physiques se voit donc attribuer aléatoirement un type. Cela créer des groupes de machines physiques de mêmes types qui ne sont pas remis en question entre deux exécutions d'un même algorithme, ou entre deux utilisations d'algorithmes différents.

Type	0	1	2	3	4
Hétérogénéité (%)	-20	-10	0	+10	+20

Table 6.1 – Chaque type de machine physique correspond à un pourcentage d'hétérogénéité de puissance par rapport au modèle de base utilisé (Tableau 6.2), ce qui signifie qu'une machine physique délivre plus ou moins de puissance pour une capacité CPU identique. Dans ce tableau, le type 2 (0% d'hétérogénéité) a donc les mêmes caractéristiques de puissance que le modèle de base, les types 0 et 1 (respectivement -20% et -10% d'hétérogénéité) délivrent moins de puissance, les types 3 et 4 (respectivement +20% et +10% d'hétérogénéité)délivrent plus de puissance.

Site Grid'5000 de Reims						
Fréquences disponibles (GHz)		0.8	1.0	1.2	1.5	1.7
Puissance (W)	***Idle***	140	146	153	159	167
	Full	228	238	249	260	272

Table 6.2 – Fréquences disponibles sur les CPUs des machines du site Grid'5000 de Reims et leurs puissances associées, *Idle* et *Full*, correspondant respectivement à une utilisation CPU de 0% et 100%.

6.1.2 Configuration des machines virtuelles

Reprenant les caractéristiques décrites dans le Chapitre 3, les propriétés des machines virtuelles sont présentées ci dessous :

- Le nombre de machines virtuelles : 400
- Capacités (CPU en MIPS et Mémoire en Mo) des machines virtuelles : 200 / 400 / 600 / 800 (MIPS et Mo)
- Le pourcentage maximum de reconfiguration de la capacité CPU accepté par les machines virtuelles : 20% de la capacité CPU de la machine virtuelle concernée
- Nombre d'instructions à exécuter par les machines virtuelles : entre 10 et 110 fois la capacité CPU de la machine virtuelle
- Temps de migration maximum : 1 seconde

Capacités et reconfigurations

Les capacités des machines virtuelles ont été définies de manière à pouvoir être utilisées aisément à la fois par le simulateur et par les algorithmes de placement. Les capacités CPU et Mémoire des machines virtuelles ont été définies proportionnellement aux capacités des machines physiques afin d'obtenir des contraintes d'allocation égales entre la capacité CPU et la capacité Mémoire des machines physiques, lorsque la fréquence maximale CPU est utilisée. Cela amenant à un nombre d'allocations minimum et maximum possibles respectivement de 3 et de 12 sur une machine physique dans des conditions de forte consolidation.

En effet, les allocations de machines virtuelles sont contraintes à la fois par la capacité CPU (ressource *fluide*) et la capacité Mémoire (ressource *rigide*) des machines physiques. Le nombre minimum de machines virtuelles pouvant être allouées sur une machine physique est obtenu lorsqu'il s'agit de machines virtuelles ayant les capacités maximum possibles : 800MIPS et/ou 800Mo. La capacité CPU de chaque machine virtuelle peut être diminuée au minimum de 20%, donnant donc, pour les machines virtuelles ayant une capacité CPU de départ de 800MIPS, une capacité effective de 640MIPS. Alors, la contrainte de placement est la suivante : $2000 > X \times 640 \Rightarrow X < 3.125 \Rightarrow X = 3$, avec X le nombre de machines virtuelles (nombre entier) qu'il est possible d'allouer, résultant dans ce cas à un nombre maximum de 3 machines virtuelles. La contrainte se retrouve être déterminée par la capacité Mémoire si 3 machines virtuelles occupant 800Mo de Mémoire chacune doivent être allouées sur la même machine physique. Comme la capacité Mémoire des machines virtuelles n'est pas variable, l'allocation doit respecter la contrainte suivante : $2500 > X \times 800 \Rightarrow X < 3.125 \Rightarrow X = 3$, donnant donc également dans cette configuration un nombre de 3 machines virtuelles maximum.

De la même manière le nombre maximum de machines virtuelles pouvant être allouées sur une machine physique est déterminé lorsqu'il s'agit de machines virtuelles possédant les capacités les plus petites possibles : 200MIPS et/ou 200Mo. Dans cette situation, la capacité CPU initiale de 200MIPS peut être ramenée à 160MIPS lorsque le taux de reconfiguration de 20% est appliqué. Cela induit donc une contrainte sur la capacité CPU de : $2000 > X \times 160 \Rightarrow X < 12.5 \Rightarrow X = 12$, donnant un nombre maximum de machines virtuelles de 12. Concernant la mémoire, les 200Mo alloués à la machine virtuelle sont incompressibles, et la contrainte se retrouve être la suivante : $2500 > X \times 200 \Rightarrow X < 12.5 \Rightarrow X = 12$, résultant également à un nombre maximum de 12 machines virtuelles sur une machine physique.

Bien entendu, le nombre minimum de 3 machines virtuelles, n'est pas une borne inférieure absolue. Un algorithme de placement peut amener à avoir deux, une seule ou zéro machine virtuelle affectée à une machine physique. Le nombre maximum de 12 est lui par contre une réelle borne supérieure, celui-ci étant obtenu en utilisant des machines virtuelles de plus faibles capacités et des machines physiques ayant leurs capacités maximales. Il ne peut dont en aucun cas y avoir plus de 12 machines virtuelles allouées sur une seule machine physique.

La configuration 110/400 (nombre de machines physiques / nombre de machines virtuelles) a été choisie de manière à avoir, théoriquement, environ un tiers des machines physiques utilisées si toutes les machines virtuelles possédaient des capacités CPU et Mémoire les plus faibles possibles. Dans cette situation, une consolidation optimale des machines virtuelles amènerait à utiliser environ un tiers des 110 machines physiques disponibles : $400/12 \simeq 33.3$, et $\frac{1}{3} \times 110 \simeq 36.6$.

En réalité, cette situation extrême n'est jamais atteinte car les capacités CPU et Mémoire des machines virtuelles sont attribuées aléatoirement. Cela permet donc d'avoir 16 types de machines virtuelles possibles. La création des machines virtuelles génère donc un ensemble de 400 machines virtuelles ayant toutes un couple [capacité CPU ; capacité Mémoire] correspondant à un de ces 16 types possibles. Afin de toujours travailler avec le même ensemble de machines virtuelles, celui-ci ne change pas entre deux exécutions d'un même algorithme, ni entre deux utilisations d'algorithmes différents.

Temps d'exécution et temps de migration

Le nombre d'instructions à exécuter par chacune des machines virtuelles a été fixé entre 10 et 110 fois la capacité CPU de la machine virtuelle concernée. Cela permet d'obtenir des durées d'exécution de machines virtuelles variées ainsi qu'un temps de simulation raisonnable d'environ 6 minutes pour chaque algorithme. Bien sûr, ces valeurs pourraient être multipliées par dix ou cent pour arriver à des valeurs réelles d'utilisation de machines virtuelles utilisées pour des services de *Cloud Computing*. Cependant, la taille des données à analyser n'en serait que bien plus conséquente sans apporter plus de précision ou de pertinence pour l'analyse des résultats de simulations.

Cela amène, avec le jeu des reconfigurations des machines virtuelles, à un temps d'exécution d'environ 120 secondes. Connaissant cette approximation du temps d'exécution qui varie en fonction des allocations produites par les différents algorithmes de placement, le temps de migration d'une machine virtuelle a été fixé à 1 seconde au maximum. En effet, 1 seconde de temps de migration pour 120 secondes d'exécution revient environ à un temps de migration de 30 secondes pour une utilisation d'une heure, ce qui peut se rapprocher d'un réel temps de migration de machines virtuelles. Pour cela, le débit réseau a donc été fixé à 800Mo/s. Une migration de machine virtuelle n'est pas sans effet sur la consommation énergétique des machines physiques concernées. Par conséquent, la consommation énergétique d'une machine virtuelle d'une machine physique vers une autre lors des simulations a été configurée pour que celle-ci engendre une consommation énergétique identique sur les machines physiques source et destination. Un modèle plus élaboré de migration [85] de machines virtuelles aurait pu être utilisé. Ce domaine d'étude étant très complexe, cela demande de nombreuses phases de méthodologie, d'analyse, et d'évaluation totalement dédiées à celui-ci. Comme expliqué en introduction, ce chapitre a pour but d'exposer des phases d'évaluations focalisées sur l'analyse et l'interprétation de l'évolution des paramètres de QoS en fonction des algorithmes et des optimisations utilisées. C'est pourquoi le modèle simplifié de migration de machines virtuelles expliqué ci-dessus a été choisi.

6.1.3 Configuration de l'algorithme génétique

Enfin, la configuration de l'algorithme génétique pour l'allocation des machines virtuelles à chaque instant de ré-allocation est la suivante :

- Taille de la population aléatoire de départ : 1500
- Taille de la population de travail : 120
- Nombre de *mutations* : 100
- Nombre de *croisements* : 90
- Nombre fixe de générations : 600

Un des intérêts majeurs de l'utilisation de cet algorithme génétique étant de pouvoir paramétrer aisément la manière dont il optimise chacune des métriques rentrant en jeu dans sa fonction objectif, les cinq versions présentées en Section 4.4.8 sont utilisées, ainsi qu'une sixième intégrant une métrique d'optimisation du nombre de migrations. Les configurations d'optimisations (valeurs des coefficients ap-

pliqués à chacune des métriques) de ces six versions différentes de l'algorithme génétique sont résumées en Tableau 6.3.

Nom GA	Coefficients appliqués aux métriques				
	Énergie	Temps de réponse	Robustesse	Dynamisme	Nombre de migrations
GA_All	1	1	1	1	0
GA_All_Mig	1	1	1	1	1
GA_Energy	1	0	0	0	0
GA_RespT	0	1	0	0	0
GA_Rob	0	0	1	0	0
GA_Dyn	0	0	0	1	0

Table 6.3 – Tableau récapitulatif des différentes versions de l'algorithme génétique associées à leurs coefficients d'optimisation de chacune des métriques de QoS.

Contrairement au Tableau 4.1 présenté dans le Chapitre 4, ce Tableau 6.3 intègre le coefficient d'optimisation de la métrique supplémentaire (nombre de migrations) incluse dans la version GA_All_Mig.

6.1.4 Utilisation des algorithmes de placement

En ce qui concerne les algorithmes de placement, le BFS et RR ont été directement implémentés dans le simulateur, contrairement à l'algorithme génétique qui est un programme C++ indépendant. Cela conduit à une petite différence dans la manière de faire appel aux algorithmes au cours des simulations. En effet, lors d'une simulation un appel périodique est fait aux algorithmes de placement toutes les 20 secondes, retournant un nouveau placement à mettre en place.

Pour les algorithmes de BFS et RR, cet appel est fait en interne, c'est à dire que c'est le simulateur qui exécute directement les algorithmes de placement, récupère les résultats, et poursuit la simulation en y intégrant les migrations de machines virtuelles à effectuer. Pour le GA, qui est un programme totalement indépendant du simulateur, la façon de procéder est un peu différente. Alors, à chaque instant t de ré-allocation, CloudSim lance l'exécutable du GA en dehors du simulateur en lui donnant tous les paramètres dont il a besoin pour trouver une nouvelle solution de placement. La configuration à chaque instant de ré-allocation est donc sauvegardée et contient :

- Les machines virtuelles encore en cours d'exécution
- Le nombre d'instructions restantes à exécuter de chaque machine virtuelle

Ces paramètres sont donnés en entrée du GA, puis le simulateur attend la fin de l'exécution du GA avant de pouvoir récupérer le nouveau placement à mettre en place. Celui-ci est transmis par fichiers au simulateur ce qui lui permet de programmer les migrations des machines virtuelles dont l'allocation a changé. L'exécution de la simulation est alors relancée, les migrations prévues sont effectuées par le simulateur, et la simulation se déroule jusqu'au prochain moment de ré-allocation (20 secondes plus tard). Ainsi de suite, jusqu'à ce que toutes les machines virtuelles aient fini leur exécution. Ces ré-allocations périodiques permettent d'une part d'améliorer le placement en tenant compte des machines virtuelles qui

ont terminé leur exécution, et permet d'un autre côté d'analyser l'effet des ré-allocations sur les différentes métriques de qualité de service dont les algorithmes tiennent compte.

Dans la section suivante 6.2, les résultats des simulations utilisant le principe de ré-allocations périodiques des différents algorithmes de placement sont nommés : BFS-ReAlloc , RR-ReAlloc et GA-ReAlloc. Des simulations ont également été réalisées sans ré-allocation en simulant uniquement le placement de départ obtenu à $t = 0$. Ces simulations sont simplement nommées : BFS, RR et GA.

Utilisation du DVFS

Dans toutes les simulations présentées dans la section suivante, le DVFS implémenté au sein de CloudSim a été utilisé dans le mode *UserSpace* (définition en Section 2.3.1). Pour les deux algorithmes gloutons directement implémentés dans CloudSim, la fréquence la plus basse possible est attribuée au CPU des machines physiques en tenant compte évidemment de la capacité occupée par les machines virtuelles allouées à celles-ci et du taux de reconfiguration à appliquer aux machines virtuelles (si celui-ci ne dépasse pas les 20% maximum autorisés). Pour les simulations utilisant les allocations générées par l'algorithme génétique, les attributions de fréquences CPU des machines physiques et des taux de reconfiguration des machines virtuelles sont directement appliqués en accord avec les valeurs correspondant aux allocations générées par le GA. Celles-ci sont passées comme paramètres d'entrée au simulateur lors de la première allocation à $t = 0$ pour le démarrage de la simulation, puis de la même manière à chaque fois que le GA est appelé (à chaque instant de ré-allocation) jusqu'à la terminaison de celle-ci.

La section suivante présente et critique l'ensemble des résultats des simulations utilisant toutes les configurations introduites ci-dessus.

6.2 Simulations

Cette section expose les résultats de simulation qui sont illustrés par de nombreuses figures montrant l'évolution et les valeurs des différents paramètres de QoS au cours du temps. En plus des comparaisons des quatre paramètres de qualité de service mesurés, une courbe du nombre de machines physiques utilisées et un histogramme du nombre de migrations de machines virtuelles déclenchées par chaque algorithme lors des ré-allocations sont proposés. Pour chaque algorithme, deux simulations sont effectuées. La première déroule uniquement l'allocation obtenue à $t = 0$, la seconde utilise cette même allocation à $t = 0$ puis effectue des ré-allocations toutes les 20 secondes. Par conséquent, entre $t = 0$ et $t = 20$ les courbes sont totalement identiques.

Aussi, des simulations utilisant les quatre autres versions de l'algorithme génétique ont été effectuées afin d'analyser l'effet d'une optimisation mono-objectif, de chacune des métriques, au cours du temps. Ces simulations utilisent donc des ré-allocations (par le GA) avantageant une seule métrique, ce qui permet de voir les effets positifs ou négatifs sur les autres paramètres. Aussi, une version intégrant une optimisation du nombre de migrations en plus des quatre autres métriques a été évaluée. Les résultats de ces simulations sont présentés dans six sous-sections séparées.

6.2.1 Optimisation multi-objectifs équitable

Cette première section d'analyse des résultats de simulation met en jeu la version de l'algorithme génétique optimisant équitablement les quatre métriques de QoS nommée *GA_All*.

Tout d'abord, la comparaison du nombre de machines physiques utilisées (Figure 6.1) permet de constater que l'allocation donnée par le GA (et par le GA-ReAlloc) utilise les 110 machines physiques disponibles. Sachant que les GA/GA-ReAlloc optimisent toutes les métriques, celles de la robustesse et du dynamisme tendent à minimiser le nombre de machines virtuelles sur chacune des machines physiques, respectivement dans le but de minimiser le risque qu'un service soit affecté par une défaillance ainsi que d'avoir le plus grand nombre de MIPS libres sur chaque machine physique. Après le début de la simulation, le GA diminue rapidement le nombre de machines physiques utilisées et devient le meilleur sur ce paramètre, montrant que la première allocation est aussi efficace lorsque le nombre de machines virtuelles diminue. Inversement au GA, le GA-ReAlloc ré-augmente le nombre de machines physiques utilisées lorsqu'une ré-allocation est exécutée. En effet, le GA-ReAlloc tend à avoir le même comportement que le GA à $t = 0$ en optimisant toutes les métriques et reproduit ce comportement au cours des ré-allocations. Ainsi, tant que le nombre de machines virtuelles est supérieur au nombre de machines physiques, l'optimisation multi-objectifs du GA-ReAlloc amène à utiliser un grand nombre de machines physiques. Ce comportement est directement lié au nombre de machines virtuelles en cours d'exécution ainsi qu'aux quatre métriques considérées. Comme on peut le remarquer à $t = 60$ et $t = 80$, bien que ce nombre (respectivement de 216 puis 138) soit largement supérieur au nombre de machines physiques disponibles, celles-ci ne sont pas toutes utilisées par le placement du GA. RR utilise également toutes les machines physiques. Ce résultat est logique en raison du comportement intrinsèque de RR qui alloue les machines virtuelles sur les machines physiques une par une. Ce comportement peut également être facilement constater à $t = 80$ secondes, la ré-allocation de RR-ReAlloc a pour effet de répartir les machines virtuelles sur toutes les machines physiques. Enfin, à $t = 0$ les BFS/BFS-realloc sont les algorithmes qui utilisent le moins de machines physiques, avec une valeur de 90. Cela se vérifie également tout au long des simulations, à l'exception du résultat de la simulation du GA qui obtient un nombre de machines physiques bien moindre que tous les autres. Il est intéressant de noter que les ré-allocations effectuées par le BFS-ReAlloc, faites toutes les 20 secondes, ont tendance à diminuer le nombre de machines physiques utilisées ce qui est un comportent inverse au GA-ReAlloc. Cela est notamment très net à $t = 80$, où l'on peut voir que la ré-allocation de BFS-ReAlloc a vraiment aidé à avoir une meilleure consolidation des machines virtuelles. L'allure de la courbe du GA-ReAlloc est remarquable : chaque ré-allocation pousse à utiliser un nombre élevé de machines physiques, puis entre deux instants de ré-allocation ce nombre diminue très rapidement jusqu'à atteindre la valeur de la courbe du BFS-ReAlloc. Ce comportement est parfaitement visible entre $t = 20$ et $t = 40$, puis entre $t = 40$ et $t = 60$. Sur ces intervalles le nombre de machines physiques passe respectivement de 109 à 89 puis de 109 à 87.

Le graphique de robustesse (Figure 6.2) représente le nombre moyen de machines virtuelles allouées sur les machines physiques (machines vides exclues). En effet, plus ce nombre est élevé, plus le risque d'avoir un service affecté par une défaillance est grand. Cette métrique tend vraiment à avoir une allocation qui distribue les machines virtuelles sur un grand nombre de machines physiques. Cela conduit donc à un comportement parfaitement antagoniste à l'optimisation de la consommation d'énergie. En analysant le

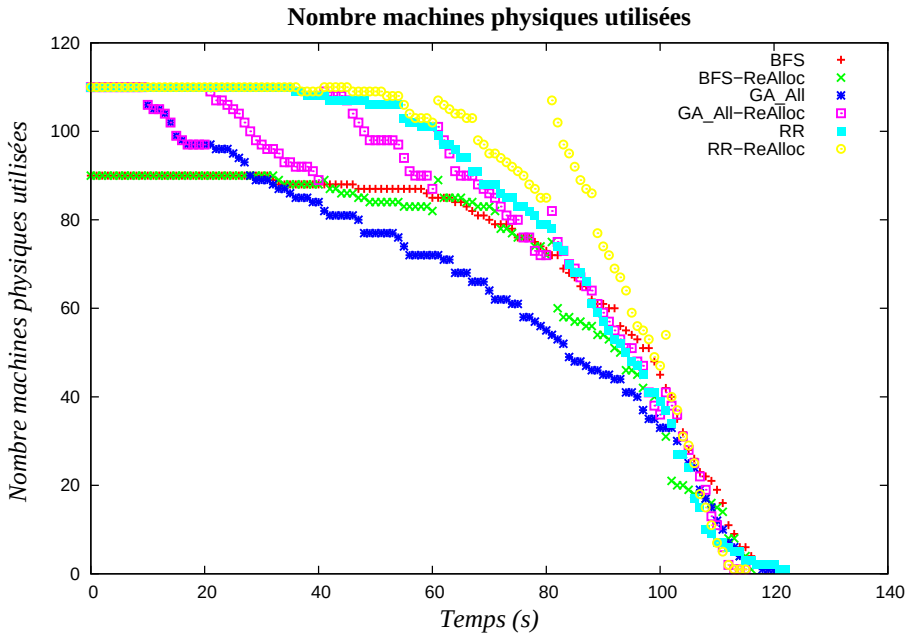


FIGURE 6.1 – Évolution du nombre de machines physiques utilisées pendant les simulations des six algorithmes.

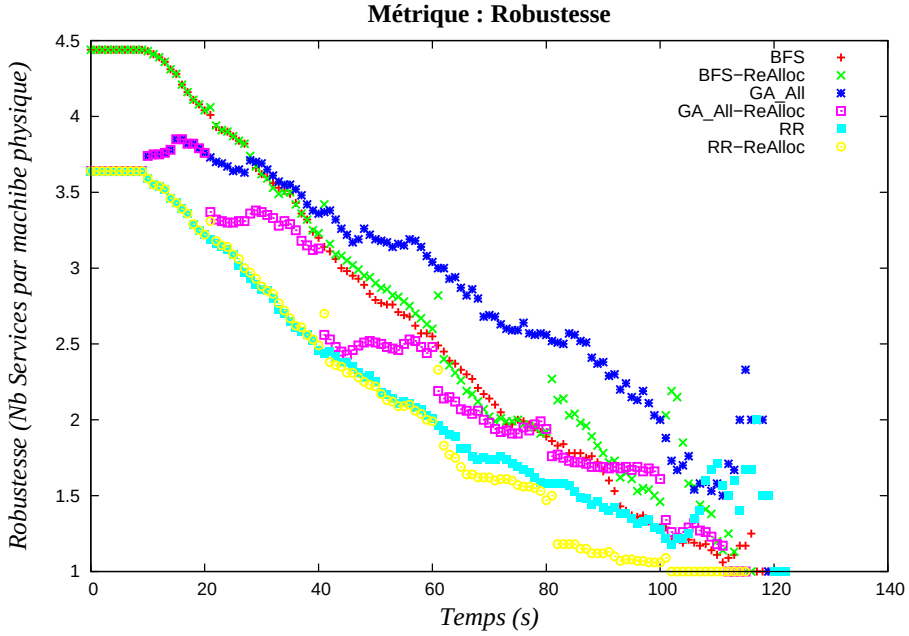


FIGURE 6.2 – Évolution de la métrique de robustesse pendant les simulations des six algorithmes.

graphique de la métrique de robustesse on peut remarquer que l'algorithme qui obtient le plus mauvais résultat sur l'ensemble de la simulation est le GA. Comme évoqué précédemment, la simulation du GA est celle qui utilise le moins de machines physiques. Ainsi, on se rend donc compte ici, avec la métrique

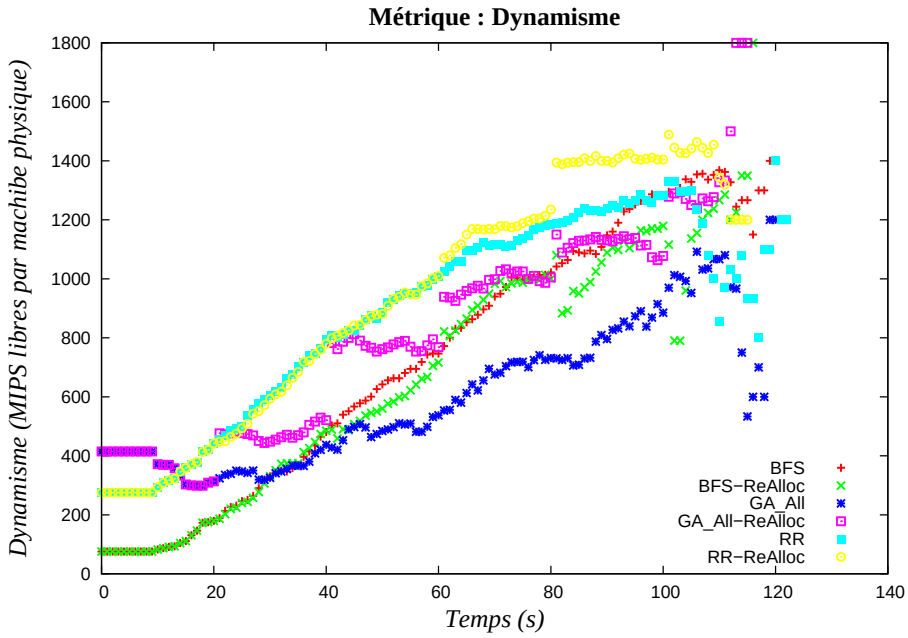


Figure 6.3 – Évolution de la métrique de dynamisme pendant les simulations des six algorithmes.

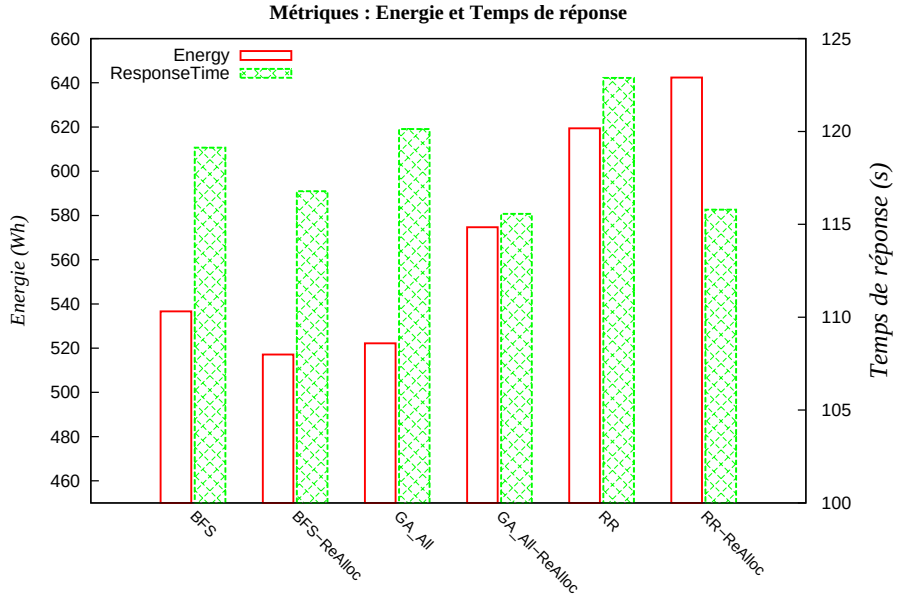


Figure 6.4 – Valeur finale des métriques énergie et temps de réponse pendant les simulations des six algorithmes. L'axe Y de gauche correspond à l'énergie (en Wh), celui de droite au temps de réponse (en seconde).

de robustesse la conséquence qui peut découler d'une minimisation du nombre de machines physiques utilisées. Le GA ayant de mauvais résultats pour la métrique de robustesse il est alors intéressant de

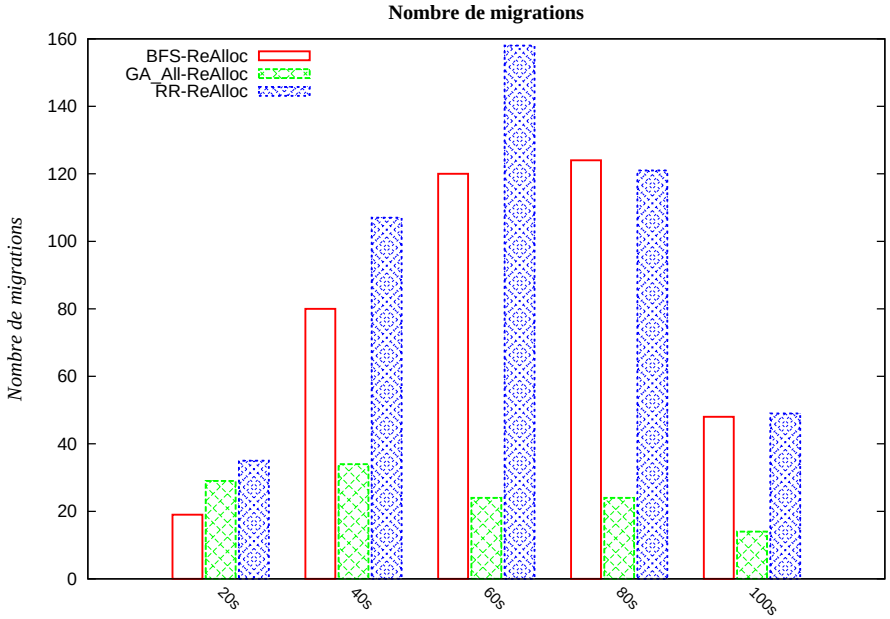


Figure 6.5 – Comparaison du nombre de migrations à chaque instant de ré-allocation engendré par les trois algorithmes. À $t = 0$, 400 machines virtuelles sont actives.

regarder la courbe correspond au GA-ReAlloc. On peut remarquer que cette courbe est la seule à avoir de grandes variations à chaque instant de ré-allocation, mettant en avant l'efficacité de l'optimisation multi-objectif sur les ré-allocations. Pour BFS et BFS-realloc, il est intéressant de noter la corrélation entre leur mauvais résultat en terme de *Robustesse* de $t = 0$ jusqu'à $t = 20$, ce qui est totalement lié à la capacité de consolidation de cet algorithme et donc au faible nombre de machines utilisées, commenté au paragraphe précédent. Il est aussi intéressant de noter (pour BFS-ReAlloc) que l'on retrouve exactement ce comportement, en voyant la valeur de robustesse augmentée à chaque instant de ré-allocation.

En ce qui concerne RR et RR-realloc, la prise en compte de cette métrique permet de constater qu'un algorithme totalement inefficace à nombreux points de vue, peut être obtenir de bon résultats si l'on ne regarde pas uniquement la consommation énergétique ou le temps de réponse. En effet, le fait de distribuer les machines virtuelles sur toutes les machines physiques produit logiquement un nombre moyen de machines virtuelles par machine physique très faible. Aussi, l'impact des ré-allocations sur cette métrique est assez bonne. Pour RR-Realloc, à chaque moment de ré-allocation une diminution assez notable du nombre moyen de services par machine physique peut être observée. Contrairement au BFS-Realloc qui a tendance à consolider nettement les machines virtuelles et donc fait diminuer la robustesse à chaque instant de ré-allocation. Il faut aussi de noter que ces deux algorithmes obtiennent de meilleurs résultats que leur version simple sans ré-allocation.

Le graphique de comparaison de résultats de simulations qui suivent, représenté par la figure 6.3, concerne la métrique de dynamisme. Ces courbes permettent tout d'abord de remarquer qu'à $t = 0$ les GA/GA-ReAlloc obtiennent un très bon résultat contrairement aux BFS/BFS-ReAlloc qui ne laissent que très peu de MIPS libres sur chacune des machines physiques qu'ils utilisent, ceci dû à leur pouvoir de

consolidation. Les bons résultat à $t = 0$ des GA/GA-ReAlloc sont bien sûr justifiés par l'optimisation multi-objectifs, mais il est également très intéressant de remarquer la différence de résultats entre les GA/GA-ReAlloc et les RR/RR-ReAlloc qui utilisent eux aussi la totalité des machines physique à $t = 0$ et donc ont théoriquement le même potentiel de résultat que les GA/GA-ReAlloc pour la métrique de dynamisme. Cette différence reflète bien une différence de comportement intrinsèque entre l'algorithme génétique et Round-Robin, à l'avantage de Round-Robin. Par la suite, plus le nombre de machines virtuelles diminue plus les courbes se rapprochent. Cependant, plusieurs remarques peuvent être faites sur ces résultats. Tout d'abord on peut constater que le GA, qui obtenait le meilleur résultat à $t = 0$ devient par la suite le plus mauvais de tous les algorithmes. Le comportement du GA-ReAlloc est remarquable. En effet les premières ré-allocations (jusqu'à $t = 40$) lui permettent de se repositionner au niveau de RR-ReAlloc, puis par la suite les ré-allocations provoquent également une amélioration mais l'écart avec RR-ReAlloc se fait de plus en plus grand : 939 contre 1009 à $t = 60$, 81 1149.88 contre 1394 à $t = 80$.

Enfin, la Figure 6.5 affiche les résultats de consommation énergétique et de temps de réponse obtenus pour chaque simulation. Globalement on peut remarquer que le GA-ReAlloc obtient des résultats intermédiaires par rapport aux BFS-ReAlloc et RR-ReAlloc, notamment concernant la consommation d'énergie . En effet, le BFS-ReAlloc très efficace en terme de consolidation de machines virtuelles minimise très bien le nombre de machine physiques utilisées comme expliqué précédemment. Cela lui permet d'obtenir une très bonne valeur de consommation d'énergie de 517.13 Wh. Son résultat de temps de réponse est également assez bon avec un résultat de 116.78s. Concernant ces deux métriques, le BFS-ReAlloc permet donc d'obtenir de bons résultats par rapport aux autres algorithmes. RR-ReAlloc affiche un temps de réponse sensiblement équivalent en 115.79s mais souffre de son inévitable tendance à utiliser un grand nombre de machines physiques qui lui donne le plus mauvais résultat énergétique avec 642.37 Wh. La simulation de l'algorithme génétique avec ré-allocations permet de conserver un temps de réponse plutôt bon de 115.56s et une consommation énergétique intermédiaire de 574.68 Wh. Il convient ici de rappeler que les résultats de l'algorithme génétique proviennent d'une optimisation multi-objectifs équitable entre les quatre métriques. Ainsi, ce résultat moins bons que celui de BFS-ReAlloc en terme de consommation d'énergie, est la conséquence de cette optimisation, qui d'un autre côté permet à l'algorithme génétique d'obtenir de bons résultats pour les autres métriques ce qui n'est pas le cas de l'algorithme Best-Fit Sorted.

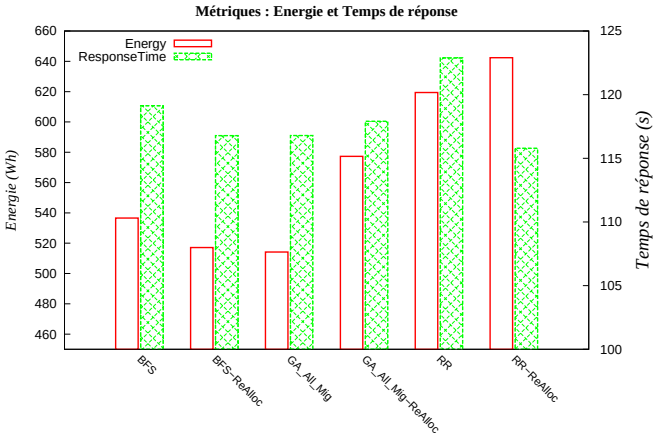
Les sections suivantes mettent en jeu les différentes versions de l'algorithme génétique présentées en Section 4.4.8, plus une version intégrant une cinquième métrique correspondant au nombre de migrations de machines virtuelles.

6.2.2 Optimisation multi-objectifs équitable avec contrainte du nombre de migrations

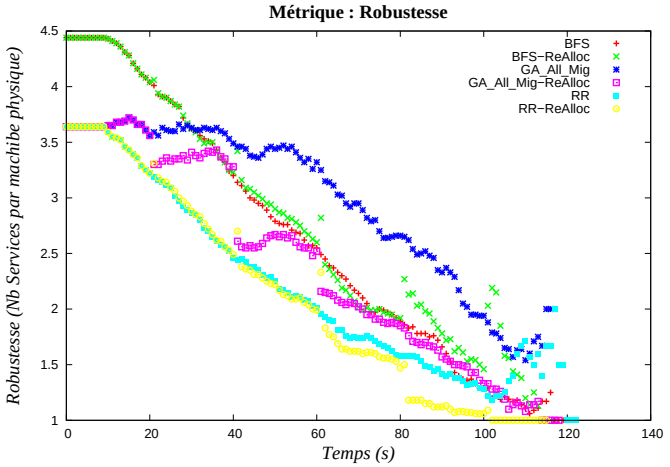
Pour cette version de l'algorithme génétique GA_All_Mig, une cinquième métrique (α_5) est utilisée afin d'intégrer l'optimisation du nombre de migrations générées à chaque pas de ré-allocation, en plus des quatre autres métriques. Une minimisation de cette cinquième métrique rentre donc en compte dans le calcul de la fonction objectif du GA. Un coefficient de 1 est appliqué à celle-ci de manière à ce que les cinq critères de la fonction objectif de l'algorithme génétique soient optimisés équitablement. L'ajout de cette métrique permet tout d'abord de voir l'influence de celle-ci sur le nombre de migrations effectuées durant

la simulation mais aussi d'analyser comment cela impacte l'évolution des autres métriques (positivement ou négativement). Le premier constat à faire est de voir que la prise en compte de l'optimisation du nombre de migrations divise environ par 2 le nombre total de celles-ci (voir Tableau 6.4). En effet, la simulation avec ré-allocation du GA_All engendre un total de 125 migrations contre 56 pour le GA_All_Mig. Il est aussi intéressant ici de se rendre compte que l'intégration de cette nouvelle métrique ne déstabilise pas complètement le comportement de l'optimisation multi-objectif. En effet, si cela avait engendré l'algorithme génétique à ne plus provoquer aucune migration de machines virtuelles, alors l'utilisation de cette métrique aurait dû directement mener à modifier (diminuer dans ce cas) la valeur du coefficient qui lui est associé. Avec cette diminution raisonnable par 2 du nombre de migrations en utilisant une valeur de coefficient égale à 1, cela permet de réguler ce paramètre de manière assez précise en faisant varier la valeur du coefficient autour de 1 (0 donnant la même que le GA_All présenté précédemment).

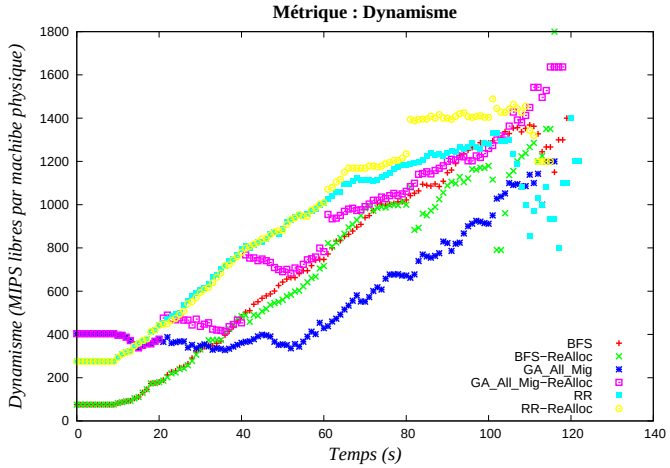
Concernant les résultats de simulations des différentes métriques obtenus avec ce GA_All_Mig, ceux-ci sont vraiment très proches, voir égaux, à ceux obtenus avec la version multi-objectifs équitable sans contrainte de migration. L'allure générale des courbes de robustesse et de dynamisme du GA-ReAlloc est vraiment similaire à celles présentées précédemment. Cela tend à déduire que le nombre de migration a une influence moindre sur le résultat global des simulations et des ordonnancements produit à chaque pas de ré-allocation. Les résultats de consommation énergétique et de temps de réponse affichent également de faibles écarts : 574.68Wh et 115.56s pour le GA_All contre 577.27Wh et 117.91s pour le GA_All_Mig. Malgré ces différences très faibles concernant ces deux métriques, cela permet tout de même de noter qu'avec un nombre réduit de migrations, la consommation énergétique ainsi que le temps de réponse sont négativement influencés par cette version GA_All_Mig. Toutes ces remarques dépendent bien sûr fortement de l'influence des migrations sur la consommation des machines physiques source et destination adoptée pour ces simulations. Il n'est pas sans dire que cette analyse pourrait être vue d'une autre façon avec un modèle de migration plus élaboré.



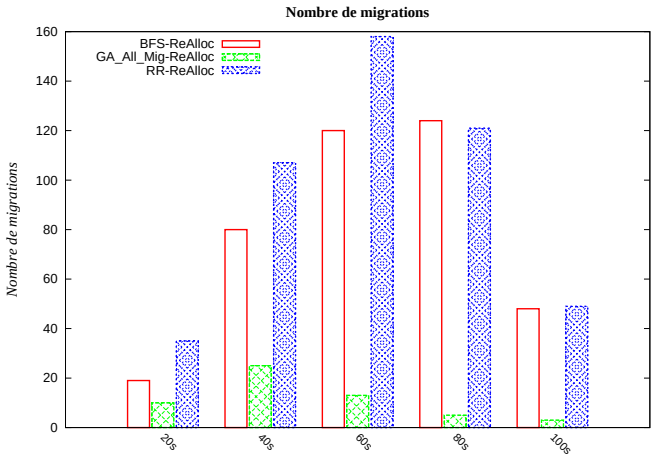
(a) Valeur finale des métriques énergie et temps de réponse. L'axe Y de gauche indique l'énergie (en Wh), celui de droite le temps de réponse (en seconde).



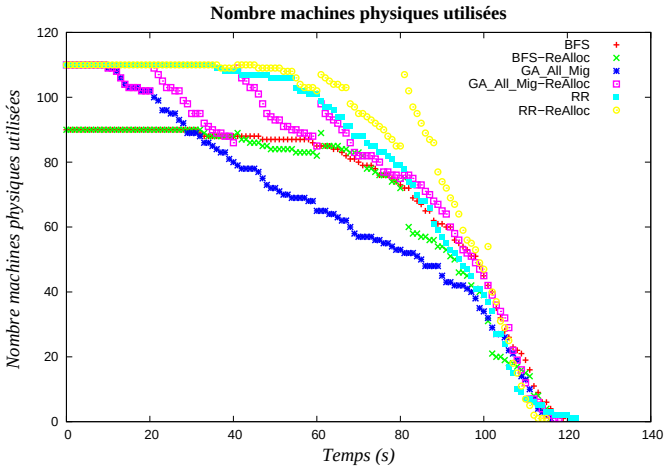
(b) Évolution de la métrique de robustesse pendant les simulations des six algorithmes.



(c) Évolution de la métrique de dynamisme pendant les simulations des six algorithmes.



(d) Comparaison du nombre de migrations à chaque instant de ré-allocation engendré par les trois algorithmes.

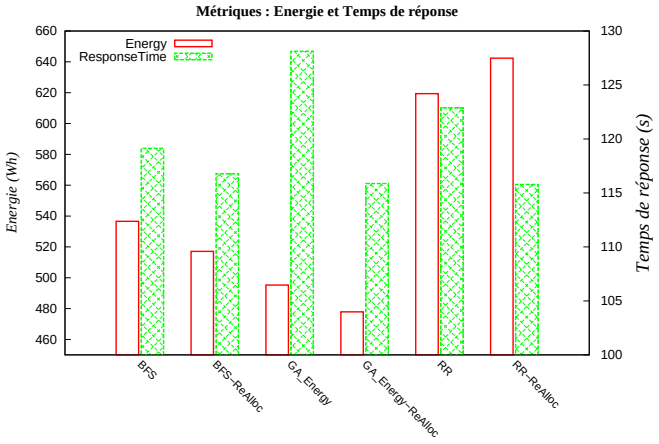


(e) Évolution du nombre de machines physiques utilisées pendant les simulations des six algorithmes.

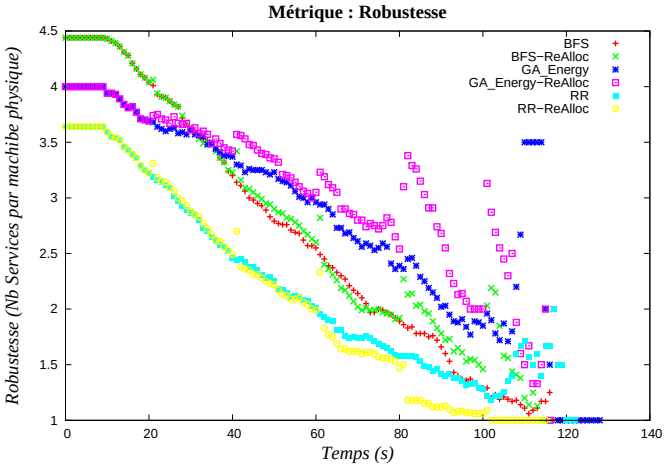
Figure 6.6 – Résultats et l'évolution des métriques de QoS observées lors des simulations. Les courbes des algorithmes gloutons sont identiques à celles présentées en Section 6.2.1, l'algorithme génétique utilisé ici optimise les quatre métriques de QoS et minimise également le nombre de migrations.

6.2.3 Mono-optimisation de l'Énergie

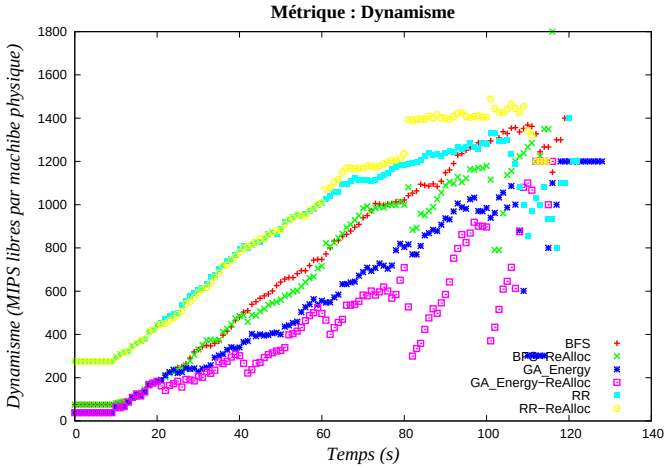
Cette section présente les résultats de simulations obtenus en utilisant la version de l'algorithme génétique GA_Energy, optimisant uniquement la métrique de consommation d'énergie (Figures 6.7a à 6.7e). Les autres métriques ne sont donc pas prises en compte dans la fonction objectif de cette version de l'algorithme génétique. Ainsi, leurs valeurs dépendent uniquement de l'optimisation de la métrique énergie. Les résultats de simulations montrent logiquement un résultat en terme de consommation d'énergie du GA-ReAlloc nettement meilleur que tous les autres algorithmes, et également largement inférieur que celui de la simulation utilisant le *GA_All*. En effet, la simulation avec ré-allocation du GA_Energy obtient une valeur de consommation énergétique de 495.27 Wh, la simulation avec ré-allocation du GA_Energy est la moins consommatrice de toutes avec une valeur de 477.89 Wh contre 517.13 Wh pour le BFS-ReAlloc déjà très performant sur cette métrique et 574.68 Wh pour la simulation avec ré-allocation du GA_All. L'analyse de ces résultat ne s'arrête pas à la comparaison de la métrique de consommation énergétique. En effet, ces résultats de simulations permettent également de regarder l'impact de l'optimisation de cette métrique sur les autres. L'influence de cette optimisation sur les métriques de robustesse et de dynamisme est très nette. Pour ces deux métriques les résultats obtenus en optimisant uniquement la métrique de consommation d'énergie sont les plus mauvais parmi les simulations des six algorithmes. Une première chose à remarquer est que ces tendances de résultats suivent les résultats d'optimisation par le GA à la fin du Chapitre 4 (en Section 4.4.8). Les courbes d'optimisations présentées dans cette section permettent de retrouver l'influence très négative de l'optimisation de l'énergie par rapport aux métriques de robustesse (Figure 4.7) et du dynamisme (Figure 4.8). La dégradation de ces métriques s'expliquent par la forte consolidation des machines virtuelles sur les machines physiques qu'implique l'optimisation de cette métrique, afin de minimiser le nombre de machines physiques allumées (clairement visible sur la Figure 6.7e). Cela engendre inévitablement une augmentation du nombre moyen de machines virtuelles par machine physique et donc une dégradation directe de la métrique de robustesse. La métrique de dynamisme est également affectée par cette consolidation et la minimisation du nombre de machines physiques allumées. La valeur de cette métrique est influencée par le ratio entre le nombre machines virtuelles à allouer et le nombre de machines physique utilisées. Ainsi, le fait d'allouer un certain nombre de machines virtuelles sur un nombre réduit de machines physiques tend donc à avoir une moyenne de capacité CPU libre (dynamisme) amoindrie par rapport à une allocation du même nombre de machines virtuelles sur un nombre de machines physiques plus important. Enfin, il est intéressant de noter que le temps de réponse n'est pas dégradé par rapport à la valeur obtenue par la simulation avec ré-allocation utilisant la version *GA_All* de l'algorithme génétique. En effet, le temps de réponse du GA-ReAlloc étudié ici est de 115.87s contre 115.56s pour le GA_All.



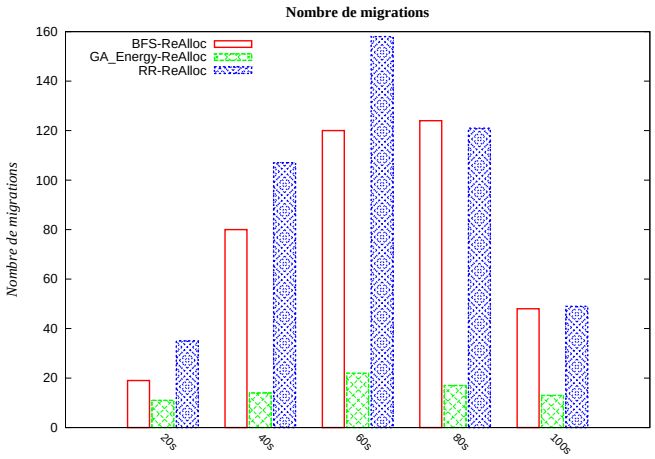
(a) Valeur finale des métriques énergie et temps de réponse. L'axe Y de gauche indique l'énergie (en Wh), celui de droite le temps de réponse (en seconde).



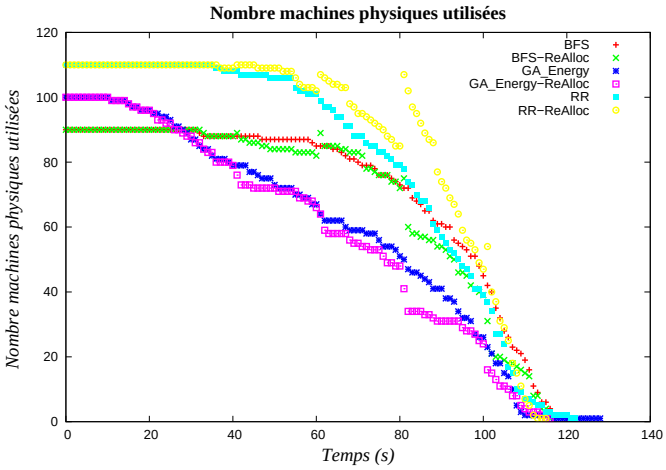
(b) Évolution de la métrique de robustesse pendant les simulations des six algorithmes.



(c) Évolution de la métrique de dynamisme pendant les simulations des six algorithmes.



(d) Comparaison du nombre de migrations à chaque instant de ré-allocation engendré par les trois algorithmes.

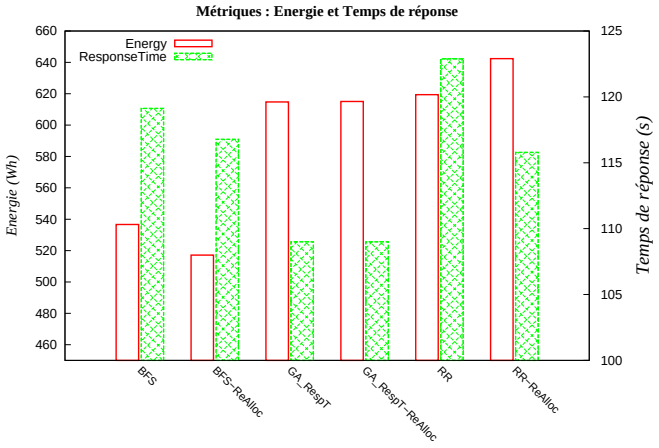


(e) Évolution du nombre de machines physiques utilisées pendant les simulations des six algorithmes.

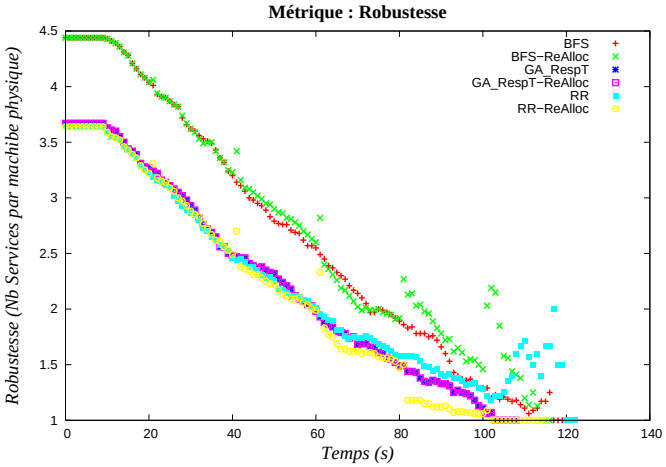
Figure 6.7 – Résultats et l'évolution des métriques de QoS observées lors des simulations. Les courbes des algorithmes gloutons sont identiques à celles présentées en Section 6.2.1, l'algorithme génétique utilisé ici optimise uniquement la métrique de consommation d'énergie.

6.2.4 Mono-optimisation du Temps de réponse

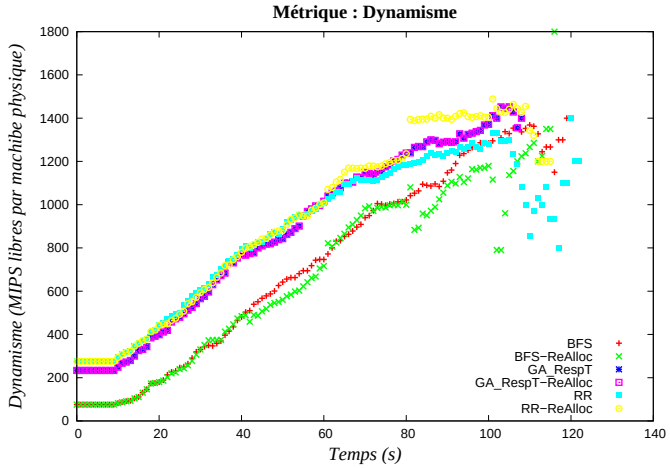
Les figures suivantes (Figures 6.8a à 6.8e) illustrent les résultats des simulations incluant la version de l'algorithme génétique dédié à l'optimisation unique de la réponse du temps de réponse. La première constatation à faire est qu'aucune migration n'a été effectuée pendant la simulation. Cela signifie donc qu'à chaque instant de ré-allocation la solution donnée par la GA, qui optimise au maximum le temps de réponse, était exactement la même que la solution de départ utilisée à $t = 0$. Logiquement, les courbes du GA_RespT et du GA_RespT-ReAlloc sont exactement les mêmes sur les figures 6.8b et 6.8c. Les résultats pour ces deux métriques sont plutôt très bons. En effet, cette version GA_RespT de l'algorithme génétique obtient pendant quasiment toute la durée de simulateur les mêmes résultats de robustesse et de dynamisme que RR et RR-ReAlloc, qui comme expliqué en Section 4.4.8 sont très performants dans ces deux métriques de QoS. Plus précisément, tant que $t < 60$ les GA sont très proches et à peine moins bon que les RR/RR-ReAlloc. Entre $t = 60$ et $t = 80$, les GA obtiennent vraiment exactement les mêmes résultats que RR et RR-ReAlloc, puis suite à la ré-allocation effectuée à $t = 80$ RR-ReAlloc devient nettement meilleur et les courbes des GA se retrouvent juste en dessous de celle de RR. Ces résultats sont expliqués par le fait que pour optimiser le temps de réponse total, aucune machine virtuelle ne doit être ralentie durant sont exécution, et donc que les reconfigurations de machines virtuelles sont à proscrire (du moins pour les machines virtuelles ayant un temps d'exécution assez long). Ce comportement est exactement celui que l'on peut observer sur ces différentes courbes. Limiter au maximum le nombre de reconfigurations de machines virtuelles rend la consolidation de celles-ci plus difficile à obtenir et est logiquement bien moins efficace. Ceci provoque donc l'utilisation d'un grand nombre de machines physiques durant toute la durée de simulation (visible en Figure 6.8e). Évidemment le prix à payer pour une telle configuration se constate en Figure 6.8a, sur laquelle les résultats en terme de consommation d'énergie sont sans appel. Les valeurs de consommation énergétiques font ressortir pour les GA_RespT d'assez mauvaises valeurs : 614.8 Wh et 615.07 Wh (pour le GA_RespT-ReAlloc). Ce résultat se place environ entres les résultats du RR-ReAlloc de 642.37 Wh et du GA_All-ReAlloc de 574.68 Wh, mais évidemment très loin de la valeur du GA_Energy-ReAlloc de 477.89 Wh. Bien sûr les résultats de temps de réponse correspondent à la meilleure valeur possible de 109 secondes, que soit pour le GA_RespT ou le GA_RespT-ReAlloc. Après ces commentaires, il apparaît que l'optimisation de cette métrique de temps de réponse à beaucoup d'impact sur les autres métriques : de façon positive pour la robustesse et le dynamisme, mais de manière très négative pour l'énergie. Une telle configuration d'optimisation n'est donc envisageable que pour un système très peu chargé, qui limitera tout de même la consommation totale du *data center*.



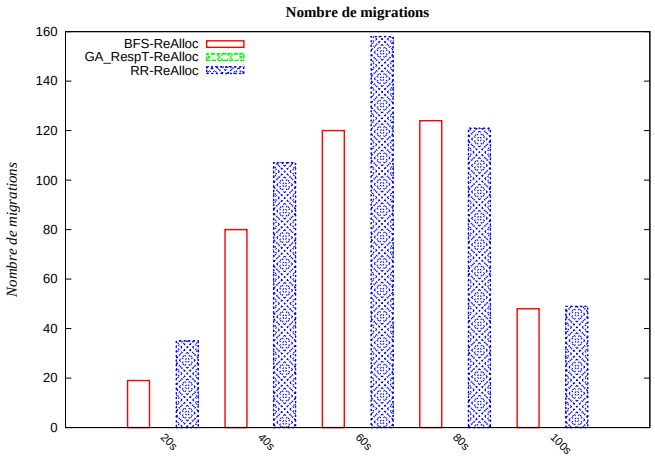
(a) Valeur finale des métriques énergie et temps de réponse. L'axe Y de gauche indique l'énergie (en Wh), celui de droite le temps de réponse (en seconde).



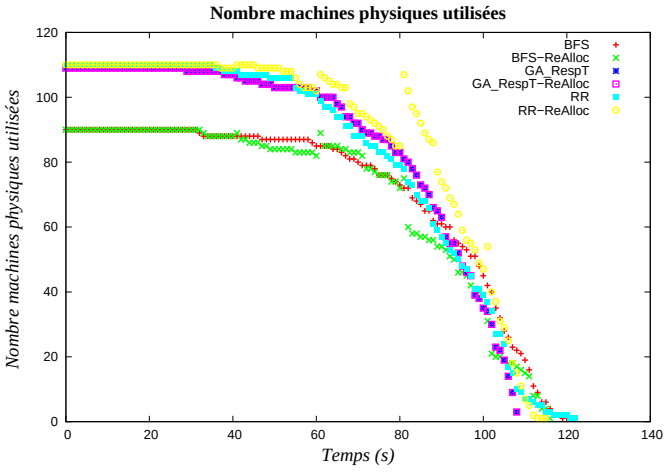
(b) Évolution de la métrique de robustesse pendant les simulations des six algorithmes.



(c) Évolution de la métrique de dynamisme pendant les simulations des six algorithmes.



(d) Comparaison du nombre de migrations à chaque instant de ré-allocation engendré par les trois algorithmes.

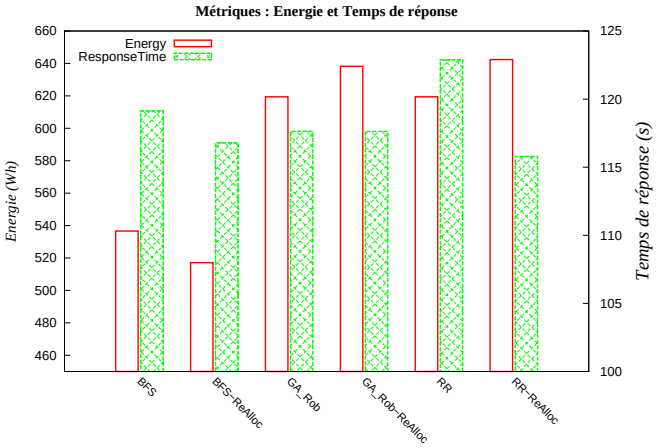


(e) Évolution du nombre de machines physiques utilisées pendant les simulations des six algorithmes.

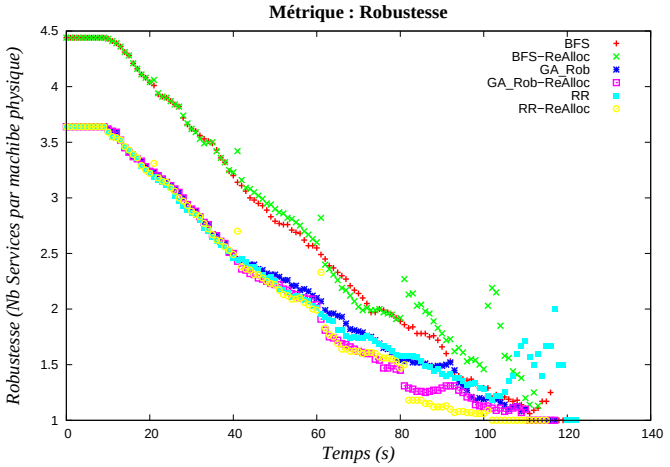
Figure 6.8 – Résultats et l'évolution des métriques de QoS observées lors des simulations. Les courbes des algorithmes gloutons sont identiques à celles présentées en Section 6.2.1, l'algorithme génétique utilisé ici optimise uniquement la métrique de temps de réponse.

6.2.5 Mono-optimisation de la Robustesse

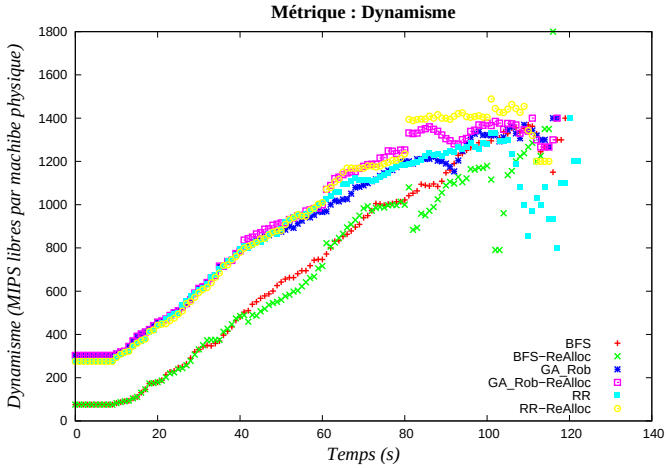
La mono-optimisation de l'algorithme génétique appliquée à la métrique de robustesse (Figure 6.9a à Figure 6.9e) fait tendre le comportement de l'algorithme génétique vers celui de Round-Robin. Comme expliqué précédemment l'allocation cyclique des machines virtuelles sur les machines physiques de Round-Robin induit inévitablement un très bon résultat en terme de robustesse. Le résultat de RR n'a pas été prouvé comme étant optimal, et ne l'est sûrement pas car différents tris des machines virtuelles pourraient sans aucun doute améliorer la robustesse résultant de ses allocations. Cependant, son comportement donne de très bons résultats, étant par moment meilleurs que les résultats donné par le GA_Rob. En effet, on peut remarquer sur la Figure 6.9b qu'à partir de $t = 80$ la courbe du GA_Rob est au-dessus de celle de RR-ReAlloc, indiquant donc une moins bonne robustesse. Ce résultat est assez étonnant étant donné le bon fonctionnement du GA dans la plupart des cas étudiés et son efficacité d'optimisation. De plus RR-ReAlloc ne donne pas le résultat optimal à cet instant ($t = 80$). À $t = 80$ RR-ReAlloc a 124 machines virtuelles à allouer, l'optimal de la métrique de robustesse serait de $\frac{124}{110}$, soit 1.13 machine virtuelle par machine physique, alors que RR-ReAlloc obtient à cet instant un résultat de 1.18 contre 1.31 pour l'algorithme génétique (qui à 1 près a le même nombre de machines virtuelles à allouer à cet instant, voir Tableau 6.5). Cette situation se poursuit jusqu'aux environs de $t = 110$s où les deux algorithmes atteignent une valeur de métrique de robustesse égale à 1 due à la diminution du nombre de machines virtuelles encore en cours d'exécution inférieur au nombre total de machines physiques disponibles. Ces différences entre les courbes de RR-ReAlloc se doivent d'être mentionnées mais sont tout de même assez infimes. Bien que le GA n'améliore pas autant la robustesse globale à $t = 80$ que la simulation de Round-Robin avec ré-allocations, le nouvel ordonnancement trouvé par le GA est tout de même très visiblement meilleur que juste avant cet instant de ré-allocation. On remarque ici la faiblesse d'une telle heuristique à pouvoir se rapprocher encore plus de l'optimal malgré une solution de départ assez bonne et ses opérateurs intrinsèques lui permettant d'explorer de nouvelles zones de l'espace de solutions. Une autre analyse est de noter que malgré la diminution progressive du nombre de machines virtuelles, la recherche de solutions aléatoires n'est pas forcément aidée dans ce cas par ce nombre de machines virtuelles nettement moins élevé qu'à $t = 0$. Cette remarque est aussi illustrée par la différence du nombre de machines physiques utilisées par ces deux algorithmes à $t = 80$. Quant aux valeurs des autres métriques obtenus par ce GA_Rob, les résultats en terme de consommation énergétique sont identiques à ceux des deux simulations de Round-Robin, malgré un temps d'exécution plus long pour le GA-ReAlloc par rapport au RR-ReAlloc, mais ce dernier effectue un nombre de migrations bien plus élevé ce qui explique l'équilibre de ce résultat énergétique.



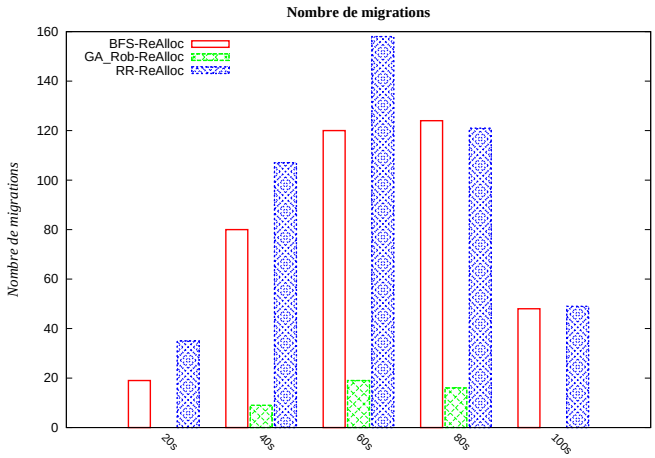
(a) Valeur finale des métriques énergie et temps de réponse. L'axe Y de gauche indique l'énergie (en Wh), celui de droite le temps de réponse (en seconde).



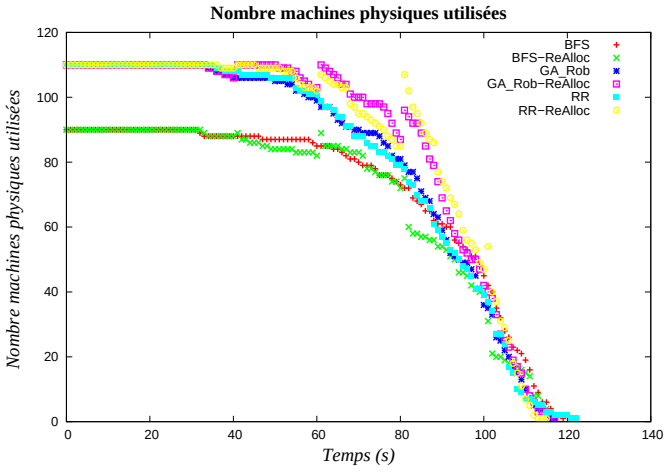
(b) Évolution de la métrique de robustesse pendant les simulations des six algorithmes.



(c) Évolution de la métrique de dynamisme pendant les simulations des six algorithmes.



(d) Comparaison du nombre de migrations à chaque instant de ré-allocation engendré par les trois algorithmes.



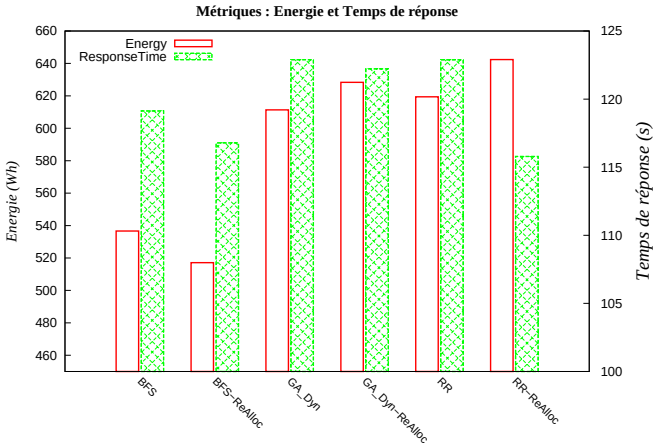
(e) Évolution du nombre de machines physiques utilisées pendant les simulations des six algorithmes.

Figure 6.9 – Résultats et l'évolution des métriques de QoS observées lors des simulations. Les courbes des algorithmes gloutons sont identiques à celles présentées en Section 6.2.1, l'algorithme génétique utilisé ici optimise uniquement la métrique de robustesse.

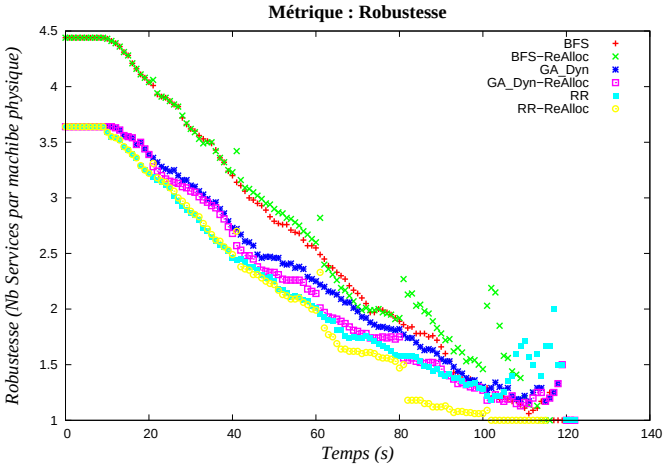
6.2.6 Mono-optimisation du Dynamisme

Les Figures 6.10a à 6.10e affichent les résultats des simulations utilisant la version de l'algorithme génétique GA_Dyn. En terme de dynamisme, la simulation du GA_Dyn donne de très bons résultats, en étant meilleur que RR/RR-ReAlloc jusqu'aux environs de $t = 30$ puis égalant les résultats de ceux-ci jusqu'à $t = 60$. Au même titre que pour la robustesse, Round Robin est très performant et atteint des valeurs de dynamisme très élevées. En effet, le fait de distribuer les machines virtuelles sur les machines physiques permet un nombre moyen de machines virtuelles par machine physique faible, ce qui engendre par rapport à la capacité maximum des machines physiques un taux d'utilisation de chaque machine physique assez réduit. Par conséquent, la moyenne capacité libre des machines physiques est très élevée. En regardant dans le Tableau 6.5 le nombre de machines virtuelles à chaque instant de ré-allocation, il apparait qu'à chaque instant de ré-allocation le nombre de machines virtuelles à allouer est un peu plus élevé pour le GA_Dyn que pour RR-ReAlloc. De plus, l'écart ne cesse d'augmenter tout au long de la simulation (9 à $t = 20$, 13 à $t = 80$). Ce paramètre renforce la bonne optimisation du dynamisme du GA_Dyn qui avec plus de machines à allouer se maintient au niveau de RR-ReAlloc. Il faut quand même noter la différence importante obtenue à $t = 80$. Là encore, comme expliqué dans la section précédente pour la robustesse, RR-ReAlloc affiche un résultat nettement supérieur au GA_Dyn. Bien que le nombre de machines virtuelles à allouer pour le GA_Dyn soit plus élevé à cet instant (137 contre 124 pour RR-ReAlloc), cette différence de dynamisme se retrouve surtout dans le nombre de machines physiques utilisées. RR-ReAlloc utilise de façon logique les 110 machines disponibles, alors que le GA_Dyn donne un placement sur 92 machines conduisant à une valeur de dynamisme bien moindre. Cette différence très marquée à cet instant de simulation est très similaire à celle reportée pour le GA_Rob dans la section précédente. En analysant les valeurs de RR-ReAlloc et du GA-ReAloc, on peut remarquer une valeur vraiment élevée de 1394 pour RR-ReAlloc. Bien qu'ayant un nombre de machines virtuelles (124) à peine supérieur au nombre de machines physiques disponibles, cette valeur de dynamisme s'approche certainement de la valeur optimale à cet instant.

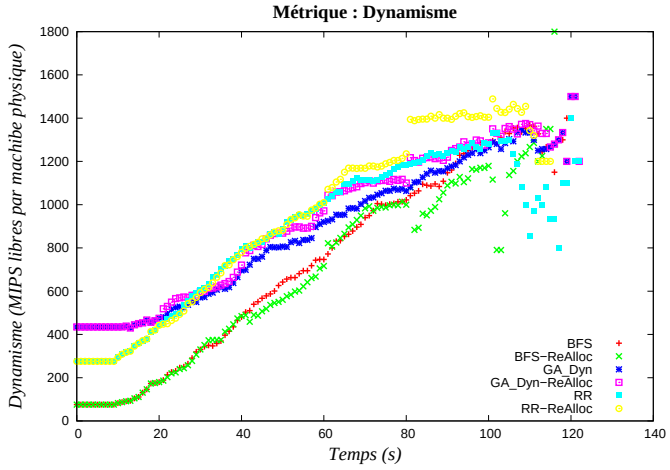
Concernant les autres métriques, cette version de l'algorithme génétique obtient des résultats assez désastreux : une consommation énergétique de 628.35Wh et un temps de réponse de 122.22s. Bien que très mauvaise cette valeur de consommation énergétique est totalement compréhensible par le fait que l'optimisation du dynamisme tend le GA à utiliser un grand nombre de machines physiques. De plus, le temps de réponse élevé dégrade aussi considérablement ce résultat. La valeur de temps de réponse obtenue signifie que le GA a effectué des reconfigurations de machines virtuelles dégradant très nettement le temps de réponse. Plus le nombre de machines virtuelles à allouer est grand plus un nombre important de celles-ci doivent être reconfigurées. En ajoutant à cela l'optimisation du dynamisme qui pousse à réduire au maximum la taille des machines virtuelles, alors ces reconfigurations engendrent un ralentissement non négligeable de l'exécution des machines virtuelles. Ainsi, ce comportement reproduit à chaque ré-allocation par le GA_Dyn récolte l'un des plus mauvais temps de réponse.



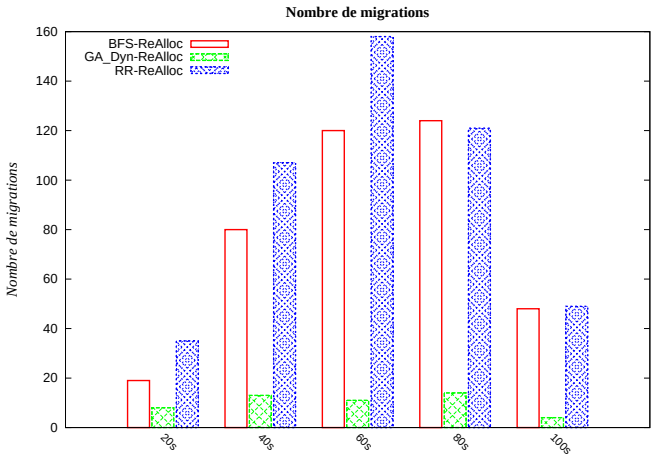
(a) Valeur finale des métriques énergie et temps de réponse. L'axe Y de gauche indique l'énergie (en Wh), celui de droite le temps de réponse (en seconde).



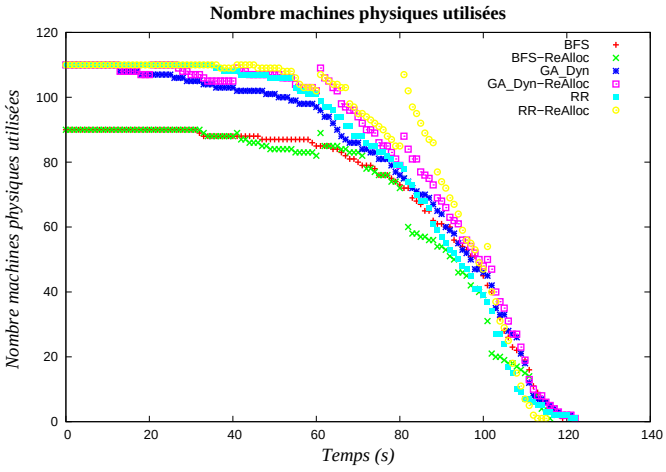
(b) Évolution de la métrique de robustesse pendant les simulations des six algorithmes.



(c) Évolution de la métrique de dynamisme pendant les simulations des six algorithmes.



(d) Comparaison du nombre de migrations à chaque instant de ré-allocation engendré par les trois algorithmes.



(e) Évolution du nombre de machines physiques utilisées pendant les simulations des six algorithmes.

Figure 6.10 – Résultats et l'évolution des métriques de QoS observées lors des simulations. Les courbes des algorithmes gloutons sont identiques à celles présentées en Section 6.2.1, l'algorithme génétique utilisé ici optimise uniquement la métrique de consommation de dynamisme.

6.2.7 Tableaux récapitulatifs

Cette section présente deux tableaux contenant des valeurs de résultats de simulations. Ces deux tableaux permettent donc d'associer les courbes et histogrammes des figures présentées dans les sections précédentes à des valeurs précises.

Le Tableau 6.4 affiche trois valeurs de résultats : la consommation d'énergie, de temps de réponse et le nombre total de migrations. Cela pour l'ensemble des simulations critiquées dans les sections précédentes.

Algorithme	**Énergie**	**Temps de réponse**	**Nombre de migrations total**
RR	619.42	122.88	⊘
RR-ReAlloc	642.37	115.79	470
BFS	536.61	119.13	⊘
BFS-ReAlloc	517.13	116.78	391
GA_All	522.21	120.13	⊘
GA_All-ReAlloc	574.68	115.56	125
GA_All_Mig	514.17	116.79	⊘
GA_All_Mig-ReAlloc	577.27	117.91	56
GA_Energy	495.27	128.12	⊘
GA_Energy-ReAlloc	477.89	115.87	77
GA_RespT	614.8	109.00	⊘
GA_RespT-ReAlloc	615.07	109.00	0
GA_Rob	619.44	117.63	⊘
GA_Rob-ReAlloc	638.26	117.63	44
GA_Dyn	611.39	122.89	⊘
GA_Dyn-ReAlloc	628.35	122.22	50

Table 6.4 – Tableau récapitulatif des valeurs des métriques d'énergie et de temps de réponse, ainsi que du nombre total de migrations engendrés lors des simulations de chaque algorithme.

Le Tableau 6.5 indique le nombre exact de quatre paramètres à chaque instant de ré-allocation (une colonne pour chaque ré-allocation) :

- Le nombre de machines virtuelles encore en cours d'exécution (**Nb VM**)
- Le nombre de migrations effectuées (**Nb Mig**)
- Le nombre de reconfigurations de machines virtuelles (**Nb Reconf**)
- Le nombre de machines physiques utilisées (**Nb PM**)

En plus des graphiques proposés précédemment, ce tableau affiche les valeurs exactes du nombre de migrations et du nombre de machines physiques utilisées. Cela permet de relier les courbes à des valeurs réelles mais surtout de quantifier leur évolutions au cours des simulations. Deux autres paramètres dont l'évolution n'a pas été proposée par graphique sont intégrés à ce tableau. Cela fournit des informations

complémentaires sur le déroulement des simulations et permet une meilleure analyse et compréhension des résultats de celles-ci en fonction des algorithmes de placement utilisés. En effet, le nombre de machines virtuelles à allouer est un paramètre prépondérant pour l'allocation de celles-ci par les différents algorithmes de placement. Ces valeurs aident donc également l'analyse des courbes des autres métriques. Le nombre de reconfigurations engendré à chaque instant de ré-allocation est également un paramètre très intéressant à analyser étant donnée son influence directe sur le temps de réponse, et la possibilité de consolider plus ou moins les machines virtuelles encore en cours d'exécution.

Algorithmes	Paramètres	Instants de ré-allocation (en seconde)						
		t = 0	**t = 20**	**t = 40**	**t = 60**	**t = 80**	**t = 100**	**t = 120**
RR-ReAlloc	Nb VM	400	354	267	202	124	49	⊘
	Nb Mig	0	35	107	158	121	49	⊘
	Nb Reconf	0						⊘
	Nb PM	0						⊘
BFS-ReAlloc	Nb VM	400	364	282	212	137	54	⊘
	Nb Mig	0	19	80	120	124	48	⊘
	Nb Reconf	0						⊘
	Nb PM	0						⊘
GA_All-ReAlloc	Nb VM	400	365	279	216	138	55	⊘
	Nb Mig	0	29	34	24	24	14	⊘
	Nb Reconf	324	200	26	3	0	0	⊘
	Nb PM	110	110	110	102	77	39	⊘
GA_All_Mig-ReAlloc	Nb VM	400	363	280	213	139	57	⊘
	Nb Mig	0	10	25	13	5	3	⊘
	Nb Reconf	320	209	45	9	3	0	⊘
	Nb PM	110	110	110	98	78	43	⊘
GA_Energy-ReAlloc	Nb VM	400	354	266	198	119	47	⊘
	Nb Mig	0	11	14	22	17	13	⊘
	Nb Reconf	60	21	8	3	3	3	⊘
	Nb PM	100	94	74	59	34	15	⊘
GA_RespT-ReAlloc	Nb VM	400	356	265	202	124	39	⊘
	Nb Mig	0	0	0	0	0	0	⊘
	Nb Reconf	125	77	20	4	0	0	⊘
	Nb PM	109	109	107	102	83	35	⊘
GA_Rob-ReAlloc	Nb VM	400	356	265	209	125	45	⊘
	Nb Mig	0	0	9	19	16	0	⊘
	Nb Reconf	187	122	34	12	3	0	⊘
	Nb PM	110	110	110	110	96	40	⊘
GA_Dyn-ReAlloc	Nb VM	400	363	281	218	137	61	2
	Nb Mig	0	8	13	11	14	4	0
	Nb Reconf	339	233	75	31	6	0	0
	Nb PM	110	110	110	110	87	50	2

Table 6.5 – Évolution à chaque instant de ré-allocation des paramètres suivants : nombre de machines virtuelles (Nb VM), nombre de migrations (Nb Mig), nombre de reconfigurations de machines virtuelles (Nb Reconf) et nombre machines physiques utilisées (Nb PM)

6.3 Bilan

Ce chapitre de simulations sous contraintes de qualité de service a mis en avant les nombreux aspects d'une approche multi-critères avec l'utilisation d'algorithmes gloutons, n'ayant aucune influence sur les prises de décision de placement puis un algorithme génétique qui lui intègre une fonction d'optimisation et de décision sur ces métriques de QoS. La première section de ce chapitre a présenté toute la méthodologie adoptée pour la mise en place de ces simulations, tant au niveau de la configuration des caractéristiques des machines virtuelles et des machines physiques que du fonctionnement conjoint entre le simulateur CloudSim et les algorithmes de placement étudiés.

Un des principaux enjeux de ce chapitre était aussi de mettre en avant les intérêts et les impacts de la prise en compte de nouvelles métriques de QoS (robustesse et dynamisme), à la fois pertinentes et antagonistes avec les deux autres métriques (énergie et temps de réponse) plus communes. En effet, trouver un ensemble de métriques ayant chacune leur propre importance et générant un conflit multi-objectif intéressant sans pour autant déstabiliser totalement cette optimisation n'est pas une tâche forcément aisée. Les résultats de simulation obtenus avec l'algorithme génétique appliquant une optimisation équitable ont pu démontrer qu'aucune des ces métriques n'était totalement désavantagée par rapport aux autres et permettait ainsi d'obtenir une optimisation à la fois efficace et équilibrée. Le second objectif de ce chapitre était donc également de démontrer l'influence de chacune de ces métriques les unes par rapport aux autres. Les simulations mono-objectif effectuées avec l'utilisation de quatre configurations différentes du GA, optimisant chacune une métrique différente a mis en avant (comme introduit dans la Section 4.4.8 du chapitre précédent), un impact important de certaines métriques par rapport aux autres. En effet, par exemple, la mono-optimisation de l'énergie dévoile un impact plutôt neutre sur le temps de réponse, mais détériore fortement les résultats de robustesse et de dynamisme. De plus, l'utilisation des algorithmes gloutons a permis de mettre en avant les bons résultats obtenus par ceux-ci par rapport à certaines métriques et ainsi de faire ressortir certains de leurs avantages parfois négligés. L'exemple le plus flagrant est l'effet très positif de l'algorithme Round-Robin sur le dynamisme et la robustesse. Enfin, tous ces résultats de simulations ont également illustrer les influences des différents algorithmes et configurations d'optimisation sur l'ordonnancement général des machines virtuelles en terme d'évolution au cours du temps du nombre de migrations, nombre de reconfigurations et du nombre de machines physiques utilisées (valeurs récapitulées en Tableau 6.5).

Le chapitre qui suit conclut ce manuscrit de thèse en synthétisant les approches et les résultats constituant l'ensemble des contributions de cette thèse, présentés au cours de ces six chapitres. Aussi, une description des perspectives de recherches pouvant découler des études menées durant ce doctorat, afin à la fois de compléter et d'étendre les modèles et les méthodes adoptés pour ces travaux est proposée.

7 Conclusions et Perspectives

Sommaire

7.1 Conclusions

Le socle des études menées lors de cette thèse repose sur l'évolution incessante de l'utilisation des *data centers*, soulevant de plus en plus de diverses problématiques. Cela dû à l'omniprésence des systèmes informatiques dans les entreprises et aux envies ou besoins de millions d'utilisateurs lambda. En effet, la tendance actuelle est de faire appel à des services délocalisés, fonctionnant au sein de *data centers* répartis dans le monde entier, et mis à la disposition des nouvelles technologies. Ce nouveau paradigme qu'est le *Cloud Computing* subit lui aussi une évolution fulgurante due à l'attractivité de ses propriétés intrinsèques : utilisation à la demande, performance des ressources de calculs, mise à jours transparente et variété des services mis à disposition des entreprises et des utilisateurs particuliers. Les services de *Cloud Computing* font aujourd'hui partie de notre quotidien, ce qui n'est pas sans déplaire aux fournisseurs de services, proposant sans cesse de nouvelles possibilités d'utilisation. L'évolution continuelle du développement du *Cloud Computing* en termes de masse budgétaire, mais aussi simplement de son nombre d'utilisateurs, entraine une concurrence accrue entre les différents fournisseurs de services.

La problématique principale d'un fournisseur de services actuel est de gérer la qualité de service de l'ensemble de son infrastructure. Dans le domaine du *Cloud Computing* la qualité de service peut représenter de nombreuses propriétés et caractéristiques, de l'ensemble des ressources, bien différentes les unes des autres. La gestion de ces différents paramètres est une tâche complexe à laquelle un fournisseur de services ne peut échapper. Cela, afin d'assurer la rentabilité de son système mais également pour obtenir la

satisfaction de ses clients. En d'autres termes, un fournisseur doit résoudre la problématique suivante : *Comment assurer une qualité de service irréprochable à moindres coûts ?* La qualité de service dédiée au *Cloud Computing* regroupe bien des paramètres de domaines différents : coûts réels, sécurité, sûreté de fonctionnement, performance des couches matérielles et logicielles, etc. La gestion de la qualité de service est donc complexe de par la variété des paramètres qui la composent.

L'utilisation d'un service de *Cloud Computing* repose sur un contrat de SLA. Ce sont donc les paramètres de qualité de service contenus dans le SLA que le fournisseur doit monitorer avec précision afin de ne pas dégrader les possibilités de ses services. À ce stade, le fournisseur de services doit savoir quelles sont ses possibilités d'action sur son système et quelles sont les répercussions de celles-ci sur les paramètres de qualité de service. C'est pourquoi il est important de connaître quelles sont les réelles corrélations entre les paramètres de qualité de service influençant les SLA et les paramètres sur lesquels le fournisseur de services peut intervenir afin de gérer ses ressources. Si les influences des différents réglages possibles, sur les paramètres de qualité de service des SLA, ne sont pas clairement visibles, il devient alors impossible pour le fournisseur de services de déterminer l'impact des différentes manières d'agir sur le fonctionnement de ses ressources. De plus, afin d'avoir une bonne lisibilité des différents paramètres de qualité de service à prendre en compte, il est nécessaire de connaître leur définition, que celle-ci indique leur signification dans un environnement de *Cloud* et qu'une métrique qui puisse être mesurée leur soit associée. C'est suite à l'analyse de la valeur de cette métrique, représentant au mieux un paramètre de qualité de service de SLA, qu'un fournisseur de services sera en mesure de prendre une décision d'action. L'expressivité de cette métrique lui donne également une information sur les parties de son système qui influencent celle-ci, ce qui peut le guider dans le choix de son intervention.

La modélisation de qualité de service proposée en Chapitre 3 (Section 3.3) de ce manuscrit, classifie et définit bon nombre de paramètres de qualité de service dédiés au *Cloud Computing*. Cette modélisation est basée à la fois sur des paramètres de qualité de service incontournables, mais aussi sur des paramètres faisant réellement partie des contrats de SLA proposés par les fournisseurs de services actuels (Section 2.4). Le but de cette contribution est de proposer une large liste de paramètres, tous associés à une définition reliée au domaine du *Cloud Computing*, puis de donner des métriques pouvant attester de leur évolution. Ainsi, par l'utilisation d'un ensemble de métriques, un fournisseur de services est en mesure de contrôler le niveau de plusieurs paramètres de qualité de service. L'analyse de ces métriques lui permet d'avoir une vue d'ensemble des paramètres qu'il souhaite maîtriser, et ainsi et ainsi de décider de changements de configuration de ses ressources, s'il le juge nécessaire, afin de garantir le meilleur niveau de qualité de service possible.

La principale problématique d'un fournisseur de services actuel est de proposer une qualité de service la meilleure possible à ses utilisateurs tout en réduisant au maximum ses propres frais. Il ressort donc que les intérêts à gérer au mieux l'ensemble de la qualité de son infrastructure sont multiples. De plus, cela rassemble des paramètres de qualité de service dont les causes et les conséquences sont très diverses. D'un point de vue centré sur le rendement, le fournisseur veut diminuer au maximum l'ensemble des coûts de fonctionnement, tout en répondant aux attentes de ses utilisateurs en leur proposant une utilisation optimale des services. Avoir la possibilité de juger du niveau de plusieurs paramètres de qualité de service représente donc un réel atout. Un fournisseur peut ainsi sélectionner les paramètres qui lui semblent les plus

importants et avoir une visibilité sur leur évolution. La meilleure des situations pour lui serait de pouvoir opter pour une configuration de fonctionnement lui permettant par exemple de bonnes performances, un haut niveau de sécurité, une fiabilité accrue et des coûts réduits. Si une optimisation simultanée de l'ensemble de ces paramètres était possible, alors la principale problématique d'un fournisseur de services n'en serait plus une. En effet, chaque paramètre est intrinsèquement différent et l'ensemble des actions sur le matériel physique ou virtuel ne peuvent évidemment pas être bénéfiques pour tous les critères de qualité de service. Une optimisation simultanée de plusieurs paramètres de qualité de service ne peut se faire sans accepter d'opter pour un compromis. Lorsqu'un fournisseur adopte un compromis, cela signifie donc qu'il accepte de dégrader certains paramètres de son système et de mettre potentiellement en péril certaines clauses de son contrat SLA. En effet, une optimisation de n'importe quel paramètre de QoS a forcément des répercussions (positives ou négatives) sur d'autres paramètres de QoS qui interviennent à importance égale dans les règles de SLA. Cependant un tel compromis peut être la meilleure des solutions à adopter à un instant donné aux yeux du fournisseur de services.

Un fournisseur de services *Cloud* se retrouve donc aujourd'hui confronté à un problème d'optimisation multi-objectifs complexe.

L'objectif de la deuxième contribution de cette thèse est de démontrer comment la modélisation des paramètres de qualité de service *Cloud* peut être utilisée à des fins d'optimisation multi-critères, pourquoi la modélisation peut valoriser aussi bien un algorithme glouton qu'une méta-heuristique, et de quelle manière une approche multi-objectifs peut être utile aux fournisseurs de services. Ce sont donc quatre métriques de qualité de service qui ont été sélectionnées, permettant de confronter différents domaines : coût, performance et sûreté de fonctionnement. Cela afin de former un problème d'optimisation assimilable à une situation à laquelle un fournisseur de services *Cloud* actuel pourrait être confronté. Afin de démontrer la pertinence des métriques de qualité de service ainsi que l'intérêt de les utiliser au sein d'une approche multi-objectifs, l'algorithme génétique présenté dans ce manuscrit a été utilisé dans différentes configurations d'optimisation. Les résultats des mono-optimisations, donnant globalement une mauvaise qualité de service, ont permis de mettre en avant tour à tour chacune des métriques et obtenir ainsi un aperçu de leur influence. Un antagonisme certain entre ces métriques a pu être observé, mettant donc à rude épreuve la résolution du problème multi-critères. Bien sûr, l'intérêt majeur de l'ensemble des études menées sur cet algorithme génétique a été d'illustrer que l'utilisation d'une approche multi-critères, au service de métriques de QoS *Cloud*, permettait d'obtenir des résultats en corrélation avec les configurations d'optimisation utilisées : la configuration d'optimisation équitable entre l'ensemble des métriques donnent dans toutes les situations étudiées les meilleurs résultats (Figure 4.9), démontrant ainsi qu'il était pertinent d'utiliser une telle approche pour un fournisseur de services afin d'avantager ou non certains paramètres de QoS en fonction de l'état de son système.

Outre le challenge de trouver le meilleur compromis possible de qualité de service à un instant donné, tout fournisseur de services doit également surveiller l'évolution de son système au cours du temps afin de détecter les variations d'utilisation de ses *data centers*. Ces variations de charge des ressources de calcul sont la conséquence directe d'une évolution d'utilisation des services. Cela signifie donc pour le fournisseur que la configuration de ses ressources, choisie pour garantir au mieux une qualité de service adaptée à un certain nombre de services en fonctionnement et à un nombre d'utilisateurs, est potentiellement moins efficace lorsque ces deux paramètres ont évolué. Une surveillance des évolutions de charge de ses systèmes

ainsi qu'une remise en question des configurations matérielles et logicielles est donc primordiale, afin de garantir le niveau de qualité de service au fur et à mesure des variations du taux d'utilisation des services. Ces changements de configuration concernent principalement la gestion des fréquences de fonctionnement des processeurs des ressources de calcul, et de l'allocation des services (ré-allocation) sur l'ensemble des machines disponibles. De plus, plus le fournisseur souhaite contrôler un grand nombre de critères de qualité de service, plus des changements plus ou moins fins de configurations vont s'imposer fréquemment. En effet, ces variations de charge n'ayant pas forcément le même impact sur chacun des paramètres de qualité de service, elles amènent le fournisseur à décider d'une reconfiguration si nécessaire et, le cas échéant, de l'ampleur de celle-ci. Cela revient donc à déterminer un nouveau compromis répondant mieux aux contraintes des nouvelles situations. L'impact de la transition entre ces deux configurations n'est pas à négliger. Passer d'une allocation de services à une autre entraîne un certain nombre de processus qui peuvent être coûteux. C'est pourquoi, la prise de décision de l'instant de reconfiguration est importante, afin de minimiser au maximum le coût de celle-ci et surtout d'éviter d'effectuer deux reconfigurations consécutives (très rapprochées dans le temps) alors qu'une seule, un peu décalée, aurait suffi à rétablir un niveau de qualité de service satisfaisant.

Au vu des difficultés à exploiter un environnement réel de *Cloud* à des fins d'optimisation de qualité de service, les campagnes d'analyses mettant en jeu les différents points cruciaux des ré-allocations de services ont été réalisées par simulation. CloudSim, simulateur de *Cloud Computing* présenté en Chapitre 5, a été choisi afin d'analyser l'impact de différentes approches d'allocation de services (algorithme génétique inclus) sur l'évolution de paramètres de qualité de service au cours du temps. Afin de pouvoir simuler une exécution de services en mettant en jeu l'ensemble des caractéristiques de l'environnement considéré (Section 3.2), les caractéristiques de CloudSim ont du être revues en quelques points. CloudSim ayant l'avantage d'accepter assez aisément la conception de nouvelles fonctionnalités, plusieurs nouveaux outils ont pu y être intégrés. Tout d'abord l'ajout du DVFS, outil autorisant la gestion dynamique des fréquences des processeurs a représenté une évolution importante de ce simulateur (Section 5.2). Cet outil, intégrant cinq modes de fonctionnement différents (tels que dans le noyau Linux) présente un comportement intrinsèque d'une granularité très fine et peut permettre un gain énergétique non négligeable des machines physiques en fonction de leurs taux d'utilisation et leurs caractéristiques de puissance. D'un point de vue logiciel, la possibilité de faire varier la capacité de calcul des machines virtuelles, responsables de l'exécution des services, a aussi été ajoutée aux possibilités du simulateur, donnant plus de flexibilité lors de l'allocation des services. Enfin, l'évaluation des métriques de qualité de service est désormais intégrée au simulateur, permettant ainsi de suivre leurs variations à chaque pas de simulation.

L'utilisation d'algorithmes de placement conjointement à CloudSim a permis de mettre en place un processus de ré-allocations périodiques. Suivant l'algorithme utilisé, les nouveaux compromis de configuration matérielle et logicielle et les nouvelles solutions d'allocation de services ont chacun leurs avantages et inconvénients vis à vis des métriques de qualité de service observées. L'algorithme génétique (dans différentes configurations d'optimisation) et deux algorithmes gloutons (*Round Robin* et *Best-Fit Sorted*) ont donc été utilisés à chaque instant de ré-allocation. Ces trois algorithmes, aux comportements très différents, proposent donc des solutions d'allocation variées, ce qui a amené à illustrer leur impact sur l'évolution des métriques de qualité de service au cours du temps. Les simulations d'ordonnancement sous contraintes de qualité de service ont mis en avant diverses analyses et résultats. Tout d'abord celles-ci ont permis

de mettre en avant l'intérêt de la prise en compte de nouvelles métriques de qualité de service, antagonistes aux métriques d'énergie ou de temps de réponse. En effet, l'analyse des résultats des simulations des algorithmes gloutons a fait ressortir que certaines métriques profitaient très positivement de leurs comportements. (*Round Robin* a démontré une capacité à optimiser la métrique de robustesse bien meilleure que les autres. *Best-Fit Sorted* tend à obtenir des placements avantageant assez nettement le coût énergétique tout en ne dégradant pas trop le temps de réponse. Bien sûr, ces bons résultats sur certaines métriques ne peuvent représenter des solutions réelles d'allocation que dans des situations particulières, étant donné l'impact négatif de ces algorithmes gloutons, aux comportements très radicaux, sur les autres métriques considérées. Les simulations utilisant les versions mono-objectif de l'algorithme génétique (GA_Energy, GA_RespT, GA_Rob, GA_Dyn) font ressortir des bilans assez similaires à ceux issus des algorithmes gloutons, dans le sens où leur optimisation est mise au bénéfice que d'une seule des métriques. Le point fort de ces simulations à optimisation unique est de démontrer l'énorme potentiel de l'algorithme génétique, permettant d'obtenir quasiment à chaque fois les meilleurs résultats sur les métriques optimisées dans des conditions de charge système évoluant au cours du temps. Les simulations des différents algorithmes cités ci-dessus montrent des résultats très tranchés vis à vis de métriques de qualité de service antagonistes. La version GA_All de l'algorithme génétique, ayant démontré précédemment une qualité d'optimisation globale très intéressante, a également été utilisée pour ré-ordonnancer les services au cours des simulations. Les résultats de ces simulations rappellent à la fois la pertinence d'observer et d'optimiser simultanément différents paramètres de qualité de service pouvant s'opposer les uns aux autres. En effet, après avoir soulevé que l'optimisation d'une unique métrique ne pouvait satisfaire la qualité de service dans sa globalité, cette version multi-objectifs équitable a redémontré que son optimisation permettait d'obtenir un réel compromis sur l'ensemble des métriques au fil des ré-allocations successives. Les courbes GA_All_Re-Alloc des Figures 6.3 et 6.2 illustrent clairement que les placements proposés par cette version de l'algorithme génétique autorisent une remise à niveau des métriques de dynamisme et de robustesse à chaque instant de ré-allocation. De plus, il est très intéressant de noter la forte dégradation de ces métriques entre deux instants d'ordonnancement, ce qui ramène au premier plan l'efficacité des ré-allocations permettant de garder globalement une qualité de service acceptable (les résultats en termes de consommation d'énergie et de temps de réponse, Figure 6.4, étant eux aussi dans une bonne moyenne). Enfin, une dernière remarque globale concerne l'ensemble des résultats obtenus par simulation, qui ont remis en lumière le fait qu'aucun algorithme ne pouvait donner une solution parfaite vis à vis de toutes les métriques. Ainsi, tout choix de configuration doit inévitablement se faire parmi des compromis avantageant de différentes façons la qualité de service globale du système.

7.2 Perspectives

Les recherches menées au cours de cette thèse ont amené à faire un certain nombre de choix, tant dans les modélisations proposées que dans les outils et approches adoptés. L'aboutissement des travaux exposés dans ce manuscrit, s'intégrant dans l'environnement défini, permet de prendre du recul par rapport aux modélisations, approches et outils utilisés pour proposer diverses ouvertures de recherche.

7.2.1 Perspectives à court terme

Extensions des modèles d'architecture matérielle et logicielle

Les extensions envisageables pour les modèles d'architecture matérielle peuvent représenter des domaines recherche à part entière et ainsi apporter de vraies avancées. Tout d'abord, cela concerne l'ajout de modèles thermiques, possiblement à différents niveaux (processeurs, machine physique, *cluster*). Des recherches actuelles s'intéressent activement à ce domaine qui apporte une vision multiple : répartition de la chaleur et des points chauds dans un *cluster*, informations sur le rendement des machines physiques et détermination de celles à mettre au repos. De plus, la température de fonctionnement d'un processeur influence directement sa consommation énergétique, ce qui dégrade donc son rendement global.

Une autre évolution du modèle d'architecture matérielle, étroitement liée à celle proposée ci-dessus, est d'intégrer une modélisation des installations des systèmes de climatisation des *data centers*, aujourd'hui indispensables à leur bon fonctionnement. Cette modélisation pourrait à la fois intégrer le coût de fonctionnement des climatisations mais aussi une vision globale de leur efficacité à différents endroits d'un *data center*.

Une autre proposition concerne l'évolution du modèle de machine physique qui pourrait intégrer une hétérogénéité plus conséquente au niveau des capacités de calcul et de mémoire et du nombre de cœurs des processeurs (multi-cœurs). En mélangeant les nouvelles configurations possibles, la composition d'un *cluster* ou d'un *data center* pourrait regrouper des machines physiques aux caractéristiques plus diversifiées.

Les extensions possibles du modèle d'architecture logicielle concerne entre autres la gestion des machines virtuelles. Notamment, l'intégration de modèles plus réalistes de migration est d'un intérêt majeur. Intégrer des modèles dont la complexité se rapprocherait de celle du fonctionnement des migrations de machines virtuelles sur de réels équipements représenterait une réelle avancée.

Extensions des modèles de qualité de service

Les modèles de qualité de service établis dans cette thèse sont bien sûr sujets à de nouvelles recherches. Cela dans le but à la fois d'apporter des paramètres complémentaires à ceux présentés, mais aussi d'intégrer ou de faire évoluer les métriques par l'intermédiaire de nouvelles analyses. Cela pourrait concerner par exemple les paramètres de cryptage, indispensables à la sécurité et au fonctionnement des services de stockage, en proposant par exemple une estimation de la qualité des méthodes de cryptage proposées et un intervalle minimum entre deux renouvellements du cryptage des données, permettant au final d'estimer au bout de combien de temps la sécurité des données se dégrade et donc d'évaluer la difficulté à les protéger.

Un autre domaine de qualité de service qui peut soulever de nombreuses recherches et des avancées certaines est la gestion des sources d'énergie. En effet, l'intégration de modèles de sources d'énergie n'ayant pas les mêmes caractéristiques et n'induisant pas les mêmes contraintes (pollution/empreinte carbone/prix) pourrait engendrer nombre de questionnements et soulever de nouvelles problématiques entre la qualité de service proposée à l'utilisateur et la manière dont un fournisseur de services doit gérer le fonctionnement de son *Cloud*. Avec la prise en compte de différentes sources d'énergie, c'est en effet la gestion du *Cloud* dans son ensemble qui peut être remise en question, en imposant des études de compromis entre le rendement de celles-ci, les coûts engendrés et l'impact des décisions de gestion à prendre sur les conditions d'utilisation.

Vers un simulateur de *Cloud* plus complet

De très nombreux paramètres (physiques ou de qualité de service) sont encore ignorés dans les simulateurs actuels, représentant autant de possibilités de recherches et d'améliorations des simulateurs de *Cloud*.

L'ensemble des idées citées dans les deux sections de perspectives précédentes, pourrait faire l'objet de nouvelles fonctionnalités à ajouter aux simulateurs de *Cloud* actuels. De plus, l'évaluation d'autres paramètres de qualité de service que ceux utilisés dans cette thèse, pourrait être intégrée aux simulateurs, permettant ainsi d'étudier d'autres phénomènes d'antagonisme ou de complémentarité.

Méta-heuristiques et algorithme génétique

La première perspective d'évolution dans ce domaine serait de finaliser la version de l'algorithme génétique mettant en jeu des services composés. En effet, celle-ci pourrait permettre de se rapprocher un peu plus du réel fonctionnement des services *Cloud* actuels. Différentes topologies de services composés sont à prévoir afin d'être en mesure de représenter différents types de services. Cela pourrait aussi amener à ré-étudier différemment la configuration de l'algorithme génétique. C'est-à-dire la façon d'utiliser les différents opérateurs : mutations multiples, croisements simples, doubles ou mixtes. De plus, d'autres approches permettant de normaliser les valeurs de différentes métriques afin de rendre comparables deux valeurs de *fitness* de deux générations consécutives seraient un moyen efficace de stopper l'algorithme. Enfin, la modélisation de ce problème d'allocation au sein d'autres méta-heuristiques serait un très bon point de départ pour à la fois comparer l'évolution de l'optimisation de chacune d'entre elles, leurs comportements face aux métriques de qualité de service et en quoi les solutions obtenues sont différentes.

À la recherche de l'optimal

En liaison avec le point précédent, une comparaison avec une méthode permettant de fournir une solution optimale à une instance donnée du problème serait une perspective de recherche très enrichissante. En effet, le fait de ne pas pouvoir situer les solutions obtenues par les méta-heuristiques par rapport à l'optimal laisse un flou quant à la qualité de celles-ci. Une perspective de recherche très enrichissante serait donc de proposer une modélisation MILP (*Mixed-Integer Linear Programming*) capable de donner la solution optimale à une instance du problème. Cette perspective représente un réel enjeu de traduction de chaque propriété des modèles d'architecture de l'environnement présentés dans cette thèse, mais aussi des métriques de qualité de service (objectifs d'optimisation), en contraintes linéaires adaptées aux solveurs (comme CPLEX [35]) permettant la résolution exacte du problème d'allocation.

Apprendre pour mieux réagir

Une perspective de recherche concernant la manière d'agir sur le système au cours du temps serait d'intégrer des méthodes d'apprentissage (*Machine Learning*) de gestion de qualité de service. Utiliser ces méthodes, tenant compte de différentes échelles temporelles, des variations de charge du système au cours de celles-ci, des paramètres à avantager en fonction du taux d'utilisation et des types de services en cours d'exécution constituerait des possibilités d'actions de reconfiguration des ressources à effectuer de manières dynamique, plus intelligente et durable.

7.2.2 Perspectives à long terme

Utilisation intelligente des énergies vertes

De nouveaux objectifs environnementaux ont émergé face aux préoccupations environnementales croissantes. L'Union européenne a adopté des objectifs ambitieux, dits des "3x20" : faire passer à 20% la part des énergies renouvelables à l'échelle européenne, réduire de 20% les émissions de CO2 des pays de l'Union par rapport à 1990, accroître l'efficacité énergétique de 20%. En prenant comme hypothèse que le taux d'utilisation des *data centers*, toujours plus performants et consommant toujours plus d'énergie, continuera à augmenter très fortement à l'échelle mondiale et que le coût des énergies fossiles et nucléaire subira également une hausse qui semble inéluctable, une gestion de l'approvisionnement des *data centers* en énergie plus intelligente s'imposera. Une perspective face à ces considérations écologiques est de favoriser une utilisation des énergies vertes (solaire, photovoltaïque, éolienne, hydraulique, géothermique, etc...) pour l'alimentation des *data centers*. Cela soulève, entre autres, des réflexions autour de la manière de produire et d'utiliser ces sources d'énergie. En effet, la production de ces types d'énergie peut s'imaginer de manières différentes : locales ou distantes. De plus, l'utilisation de celles-ci est limitée par les phénomènes météorologiques ayant une influence directe sur la quantité d'énergie pouvant être générée. La production locale engendrerait un coût d'installation et de gestion pour les propriétaires des *data centers*. L'acheminement depuis des sources de production distantes engendrerait un coût supplémentaire ainsi qu'une forte incertitude sur la capacité d'énergie disponible aux moments voulus. Dans les deux cas, les gestionnaires des *data center* devront s'adapter aux variations de production et donc intégrer des politiques de gestion de consommation d'énergie encore plus fines qu'aujourd'hui. Malgré de nombreux points d'interrogation, l'utilisation de ces différents modes de production d'énergie, très en vogue de nos jours, sera l'une des évolutions indispensables du *Cloud Computing*.

Vers un Cloud distribué

La méthode d'utilisation actuelle de services *Cloud* est d'accéder à des ressources distantes, localisées dans un ou plusieurs *data centers* potentiellement géolocalisés à différents endroits du globe terrestre. Cependant, une perspective probable de l'évolution du *Cloud Computing* est de faire des ordinateurs personnels d'aujourd'hui des ressources de calcul à part entière, qui représenteraient chacune une petite partie d'un *Cloud* distribué. Chaque ordinateur personnel deviendrait donc une ressource de calcul comme une autre, pouvant héberger des services utilisés par un autre membre du *Cloud* (ou une tierce personne). En admettant qu'une telle architecture de *Cloud* voie le jour, c'est alors le concept et la définition même des SLA qui seraient à revoir. Ces derniers ne seraient plus des contrats entre un unique utilisateur et un unique fournisseur de services, mais un ensemble de clauses permettant d'attester de l'état global d'un *Cloud* formé de machines personnelles, forcément hétérogènes en de multiples points. Une autre perspective de remise en question de SLA est d'imaginer une proposition de différentes classes de SLA, intégrant des contraintes d'adhésion plus ou moins restrictives. Ainsi, des sous-ensembles de ressources seraient définis, chacun adapté à différents types de services n'imposant pas les mêmes niveaux de qualité de service.

Remettre en question le concept de fonctionnement

Depuis l'apparition des premiers *data centers* de *Cloud Computing*, le concept de fonctionnement est

basé sur l'utilisation de machines virtuelles s'exécutant au-dessus d'un hyperviseur et intégrant un système d'exploitation identique à celui utilisé sur une machine physique personnelle. De plus, à des fins notamment d'isolation, chaque machine virtuelle remplit une fonction bien précise. Ce sont donc des milliers de systèmes d'exploitation qui sont utilisés pour une unique fonction et autant de machines virtuelles à gérer. Ce principe de fonctionnement présente donc certaines lourdeurs. Une perspective d'allègement du fonctionnement des services *Cloud* serait de proposer des systèmes d'exploitation plus légers, dont les caractéristiques seraient mieux adaptées à l'exécution de ces services plus ou moins complexes. Cette évolution serait également bénéfique à l'apparition de *Cloud* distribués, hébergés sur des ordinateurs personnels dont les capacités en termes de calcul et de mémoire restent plus limitées que celles des serveurs actuels.

A

Mesures de puissance avec DVFS sur la plate-forme RECS

A.1 Présentation de la plate-forme

RECS (Figure A.1) est un prototype de plate-forme de calcul de haute efficacité énergétique. Elle intègre de nombreuses fonctionnalités de reconfiguration, des possibilités de suivi efficaces et des méthodes de contrôle permettant de suivre son état. Cela avec une granularité très fine et un *overhead* négligeable pour les ressources de calcul et de réseau.

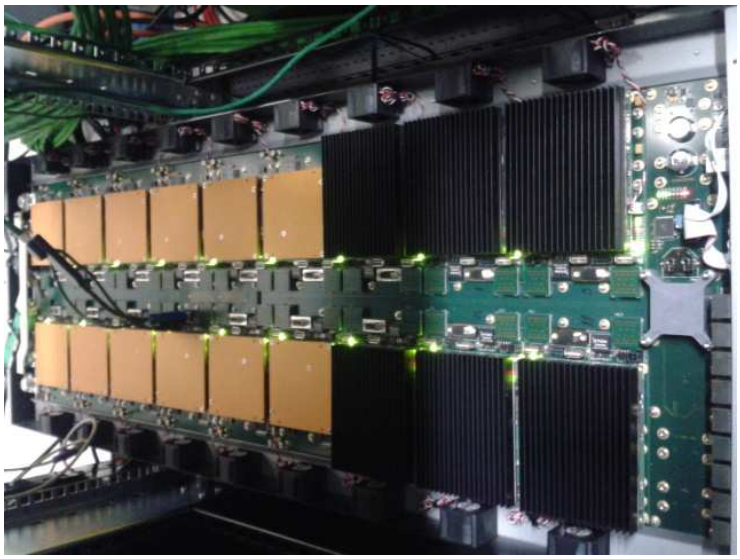


FIGURE A.1 – Photo de la plate-forme RECS de Toulouse

Le *cluster* RECS se compose de 18 modules mono-processeur, chacun d'entre eux pouvant être considéré comme une ressource de calcul individuelle. Son architecture est illustrée en Figure A.2. Cette plate-forme comporte 12 processeurs "Atom", et 6 processeurs "Intel I7". Les cartes mères sont de type "COM Express"

intégrant des modules de CPU, chacun monté sur une support permettant d'utiliser toutes les cartes "COM Express". Chaque carte de base est connectée à un panneau central arrière. Ce panneau a deux fonctions : tout d'abord il relie chaque réseau gigabit ethernet des modules CPU du *front panel* du serveur, mais permet aussi de connecter chaque micro-contrôleur au micro-contrôleur maître central.

L'outil de contrôle de la plate-forme RECS a pour but de ne pas surcharger le trafic réseau au sein de celle-ci, permet d'éviter une dépendance entre chaque nœud de calcul avec la couche de système d'exploitation et constitue une base sur laquelle de nouveaux concepts de contrôle et de surveillance peuvent être développés. Ainsi, chaque module donnent des informations sur l'état de chaque nœud de calcul, connecté à un micro-contrôleur indépendant, afin de pouvoir gérer toutes les données mesurées. Chaque nœud est équipé d'un capteur thermique et d'un capteur de consommation de courant électrique. Toutes les données des capteurs sont lues par un micro-contrôleur par nœud avant d'être récupérées par le micro-contrôleur maître. Ainsi, les surcouts potentiels causés par les mesures et les transferts de données pouvant perturber le fonctionnement d'un système à grande échelle, sont évités ce qui représente un avantage non négligeable. Cette architecture de contrôle à base de micro-contrôleurs est accessible aux utilisateurs par un port réseau dédié nécessitant une requête unique afin de récupérer toutes les informations souhaitées des nœuds de calcul. Ceci permet donc d'obtenir toutes les valeurs des capteurs de l'ensemble des 18 nœuds en une seule étape. Sans cette fonctionnalité, un utilisateur voulant par exemple analyser 10 mesures différentes sur l'ensemble des 18 nœuds aurait du effectuer 180 requêtes séparées.

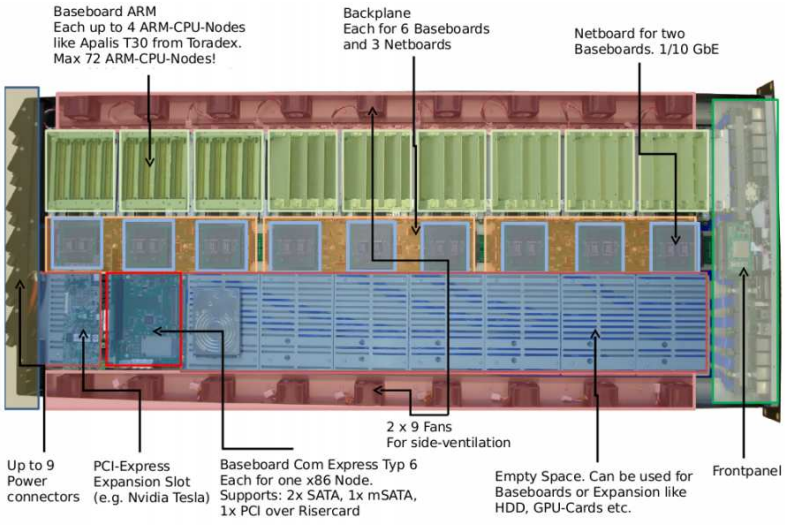


Figure A.2 – Architecture physique de la plate-forme RECS de Toulouse

A.2 Cas d'utilisation de la plate-forme

La plate-forme RECS a été utilisée afin d'effectuer des mesures de puissance sur les CPU Intel I7 (4 cœurs), dans différentes configurations de fréquence CPU. Ces expérimentations nécessitent l'activation du DVFS sur les processeurs permettant de gérer les fréquences des cœurs de chacun des CPUs. Afin d'obtenir des valeurs de mesures précises, tous les nœuds "Atom" ont été éteints, et les mesures ont été

effectuées simultanément sur les 6 processeurs “Intel I7”. Aussi, chaque configuration de fréquence CPU a été appliquée aux 6 CPUs utilisés.

A.3 Commandes de configuration et de mesures

Gouverneurs et Fréquences utilisables

Gouverneurs DVFS utilisés :

coeur 0 : performance
coeur 1 : performance
coeur 2 : performance
coeur 3 : performance

Fréquences (en Hz) disponibles (13) :
2301000 (Turbo Boost)
2300000 2200000 2100000 2000000
1900000 1800000 1700000 1600000
1500000 1400000 1300000 1200000

Configuration Adresse MAC du plogg

hcitool scan
Scanning ...
00 :80 :98 :E9 :20 :4E Plogg No Name

Connexion RFCOMM

rfcomm connect 0 00 :80 :98 :E9 :20 :4E 1
Connected /dev/rfcomm0 to 00 :80 :98 :E9 :20 :4E on channel 1
Press CTRL-C for hangup

Envoi, compilation et lancement du code micro_plogg.c

Le programme C, “micro_plogg.c” permet de récupérer les mesures du wattmètre branché sur la plate-forme. Ainsi, une mesure de puissance est récupérée toutes les secondes.

scp -P 2220 micro_plogg.c tguerout@recs.irit.fr :.
gcc micro_plogg.c -o plogg
./plogg

Récupération de la mesure de puissance :

Puissance totale de la plate-forme stockée dans : “/recs/tmp/power_plogg”.

Commande pour voir cette valeur toutes les secondes :
watch -n 1 cat /recs/tmp/power_plogg

État de la plate-forme :

Commande pour avoir une vue globale de l'état de la plate-forme :

watch -n 1 manage_recs -l

Ce qui donne :

Num	State	On/Off	Temp	Power	Type CPU
Node : 1 ;	Baseboard : Used ;	Node : Off ;	Temp : 27 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 2 ;	Baseboard : Used ;	Node : Off ;	Temp : 26 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 3 ;	Baseboard : Used ;	Node : Off ;	Temp : 26 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 4 ;	Baseboard : Used ;	Node : Off ;	Temp : 26 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 5 ;	Baseboard : Used ;	Node : Off ;	Temp : 26 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 6 ;	Baseboard : Used ;	Node : Off ;	Temp : 25 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 7 ;	Baseboard : Used ;	Node : On ;	Temp : 26 Celsius ;	Power : 11 Watt ;	Type : i7 ;
Node : 8 ;	Baseboard : Used ;	Node : On ;	Temp : 26 Celsius ;	Power : 12 Watt ;	Type : i7 ;
Node : 9 ;	Baseboard : Used ;	Node : On ;	Temp : 26 Celsius ;	Power : 12 Watt ;	Type : i7 ;
Node : 10 ;	Baseboard : Used ;	Node : Off ;	Temp : 26 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 11 ;	Baseboard : Used ;	Node : Off ;	Temp : 26 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 12 ;	Baseboard : Used ;	Node : Off ;	Temp : 26 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 13 ;	Baseboard : Used ;	Node : Off ;	Temp : 25 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 14 ;	Baseboard : Used ;	Node : Off ;	Temp : 26 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 15 ;	Baseboard : Used ;	Node : Off ;	Temp : 25 Celsius ;	Power : 0 Watt ;	Type : Atom ;
Node : 16 ;	Baseboard : Used ;	Node : On ;	Temp : 24 Celsius ;	Power : 10 Watt ;	Type : i7 ;
Node : 17 ;	Baseboard : Used ;	Node : On ;	Temp : 24 Celsius ;	Power : 11 Watt ;	Type : i7 ;
Node : 18 ;	Baseboard : Used ;	Node : On ;	Temp : 23 Celsius ;	Power : 11 Watt ;	Type : i7 ;

Table A.1 – État de la plate-forme avec la commande "manage_recs -l".

A.3.1 Tableau des mesures

Le Tableau A.2 présente la totalité des mesures effectuées. Certaines mettant en jeu différentes fréquences pour certains cœurs d'un CPU. Par exemple, la deuxième ligne du tableau "3×12 / 1×230" signifie qu'un cœur fonctionnait à la fréquence de 1.2GHz et que les trois autres cœurs fonctionnaient à la fréquence de 2.3GHz. Ainsi, plusieurs mesures ont été effectuées en mélangeant les fréquences utilisées sur chacun des cœurs.

Le wattmètre branché à la plate-forme donnant la puissance totale délivrée par celle-ci, une mesure "à vide" (tous les CPU éteints) de celle-ci avait été effectuée avant de commencer ces mesures (valeur en première ligne du tableau).

Freq (GHz)	**État CPU**	**Puissance plate-forme (W)**	**Puissance 6 CPU (W)**	**Puissance (W) 1 CPU (W)**	**StdDev (plate-forme)**
All CPUs OFF (Platform Only)	⊘	57.825	⊘	⊘	.14
1×12 3×230	Full	263.989	206.164	34.360	.14
1×12 3×230	Idle	140.270	82.445	13.740	1.87
2×12 2×230	Full	263.880	206.055	34.342	.20
2×12 2×230	Idle	140.225	82.400	13.733	1.87
3×12 1×230	Full	263.928	206.103	34.350	.14
3×12 1×230	Idle	140.137	82.312	13.718	1.83
3×12 1×231	Full	379.535	321.710	53.618	.48
4×12	Full	208.815	150.990	25.165	.14
4×12	Idle	139.964	82.139	13.689	1.99
4×16	Full	226.936	169.111	28.185	.14
4×16	Idle	140.055	82.230	13.705	1.95
4×20	Full	247.630	189.805	31.634	.30
4×20	Idle	140.167	82.342	13.723	1.90
4×21	Full	251.236	193.411	32.235	.20
4×22	Full	256.345	198.520	33.086	.14
4×230	Full	263.830	206.005	34.334	.14
4×231	Full	382.502	324.677	54.112	.20
4×231	Idle	140.777	82.952	13.825	2.83

Table A.2 – Valeur de puissance (en Watt) délivrées par les 6 CPU "Intel I7 (4 cores)" de la plate-forme RECS. La troisième colonne indique la puissance totale délivrée par l'ensemble de la plate-forme (écart-type des mesures en colonne 6). La quatrième et cinquième colonne indiquent respectivement la puissance délivrée par les 6 CPU, et par un seul CPU. Toutes les valeurs ont été calculées sur une moyenne de 120 mesures (environ 2 minutes de mesure) récupérées du wattmètre branché sur la plate-forme et donnant donc la puissance totale délivrée par celle-ci.

A.4 Validation (ou non) du modèle

Équation de la puissance délivrée par une machine physique à un instant t.

$$P^h(t) = P^h_{min}(t) + \omega^h_{f,cpu}(t)\left[P^h_{max}(t) - P^h_{min}(t)\right] \tag{A.4.1}$$

avec $\omega^h_{f,cpu}(t)$ le pourcentage d'utilisation CPU de la machine physique h

Dans l'équation A.4.1 les variables $\xi^h_{f,cpu}(t)$,$P^h_{min}(t)$ et $P^h_{max}(t)$ dépendent de la fréquence utilisée sur chacun des cœurs des CPUs "Intel I7" à un instant t. Ces variables sont donc égales à :

$$\xi^h_{f,cpu}(t) = \sum_{c=0}^{n^c} \xi^c_{f,cpu}(F_i(t))$$

$$P^h_{min}(t) = C + \left[\sum_{c=0}^{n^c} P^c_{min}(F_i(t))\right]$$

$$P^h_{max}(t) = C + \left[\sum_{c=0}^{n^c} P^c_{max}(F_i(t))\right]$$

avec C une constante désignant la puissance, considérée comme constante, délivrée par tous les autres composants.

A.4.1 Mise sous forme d'équations et de système linéaire

Désignation des variables :

$F_1 = 1.2$ GHz

$F_2 = 1.6$ GHz

$F_3 = 2.0$ GHz

$F_4 = 2.3$ GHz

Équations pour la résolution du système linéaire correspondant aux valeurs de P^h_{max}.

$$25.165 = C + 4 \times P_{max}(F_1) \tag{A.4.2}$$

$$28.185 = C + 4 \times P_{max}(F_2) \tag{A.4.3}$$

$$31.634 = C + 4 \times P_{max}(F_3) \tag{A.4.4}$$

$$34.334 = C + 4 \times P_{max}(F_4) \tag{A.4.5}$$

$$34.360 = C + 1 \times P_{max}(F_1) + 3 \times P_{max}(F_4) \tag{A.4.6}$$

$$34.342 = C + 2 \times P_{max}(F_1) + 2 \times P_{max}(F_4) \tag{A.4.7}$$

A.4.2 Validation du modèle de puissance multi-cœurs

Afin de valider le modèle de puissance multi-cœurs présenté, le système linéaire exposé ci-dessus à été résolu à l'aide de l'outil Octave.

```
./resolvRECS.sh

A =                                B =            X =

   1   4   0   0   0                 25.165          6.1370
   1   0   4   0   0                 28.185          6.1370
   1   0   0   4   0                 31.634          5.1510
   1   0   0   0   4                 34.334          5.5120
   1   1   0   0   3                 34.360          6.3743
   1   2   0   0   2                 34.342          7.5107

Temps de résolution : 0.0011 seconde(s)
```

Il convient alors de confronter les valeurs de puissance calculées avec les résultats de la résolution du système linéaire et celles obtenues par les mesures réelles effectuées sur la plate-forme.

A.4.3 Comparaison valeurs réelles et valeurs du modèle

Freq (GHz)	**État**	**Puissance RECS (W)**	**Puissance (W) Model (W)**	**Erreur (%)**
4×12	Full	25.165	26.741	5.9
4×16	Full	28.185	28.185	0
4×20	Full	31.634	31.633	0
4×230	Full	34.334	36.180	5.10
1×12 3×230	Full	34.360	33.820	-1.60
2×12 2×230	Full	34.342	31.46	-9.16
3×12 1×230	Full	34.350	29.10	-18.04

Table A.3 – Comparaison entre les valeurs mesurées sur la plate-forme et les valeurs calculées à partir du modèle, dans différentes configurations de fréquences.

A.5 Conclusion des tests effectués

Comme on peut le remarquer dans le Tableau A.3, le taux d'erreur entre les valeurs du modèle et les valeurs réelles ne dépassent pas 6% lorsque les 4 cœurs des CPUs fonctionnent à la même fréquence.

Puis, lorsque l'on effectue des mesures en utilisant des fréquences différentes sur les cœurs des CPUs, les résultats du modèle deviennent de plus en plus mauvais.

Ces résultats donnent à penser qu'il y a eu un phénomène non contrôlé dans le comportement du DVFS lors des mesures effectuées. Pour confirmer ou infirmer cela, une dernière série d'analyse du comportement du DVFS a été menée. En effet, il devient intéressant de comprendre ce qui se passe lorsqu'on effectue un changement de fréquence sur un seul cœur de CPU. Il semblerait que lorsqu'un changement de fréquence est effectué sur un seul cœur de CPU, cela affecte en réalité les 4 cœurs du CPU.

Trois possibilités sont alors possibles :

- Soit les 4 cœurs du CPU utilisent la dernière fréquence attribuée à l'un des cœurs
- Soit les 4 cœurs du CPU utilisent la fréquence la plus élevée attribuée à l'un des cœurs
- Soit les 4 cœurs du CPU utilisent la fréquence de l'un des cœurs (cpu0 ou cpu3, par exemple), et c'est alors l'ordre des changements de fréquence sur les cœurs qui prend le dessus sur la configuration attribuée

Le Tableau A.4 est le résultats de différents test de changement de fréquence sur les cœurs des CPUs, mettant en jeu les trois cas détaillés ci-dessus, afin de comprendre le comportement du DVFS sur ce type de CPU.

Freq (GHz) CPU	→	**Freq (GHz) CPU**	**État**	**Puissance (W) CPUs**	**StdDev (Plate-forme)**	**Comment**
4×230	→	1×12(cpu0) 3×230	Full	263.917	.14	Same as 4×230
4×230	→	3×230 1×12(cpu3)	Full	263.802	.17	Same as 4×230
4×22	→	1×12(cpu0) 3×22	Full	256.395	.20	Same as 4×22
1×12 3×22	→	1×12(cpu0) 3×16	Full	227.112	.17	Same as 4×16
4×12	→	3×12 1×20 (cpu3)	Full	248.952	.20	Same as 4×20

Table A.4 – Influence de l'ordre des changements de fréquence des CPUs.

En analysant les résultats de ces tests de changements de fréquence, il s'avère donc que quel que soit l'ordre ou le numéro du cœur sur lequel on applique un changement de fréquence, c'est en réalité la fréquence la plus élevée, quel que soit le numéro du cœur sur laquelle elle est appliquée, qui est attribuée aux 4 cœurs du CPU.

Ainsi, les mauvais résultats obtenus par calcul avec le modèle de puissance multi-cœurs, lorsque différentes fréquences sont attribuées aux différents cœurs d'un CPU, sont compréhensibles au vu des résultats des tests de l'influence de l'ordre d'attribution des fréquences (Tableau A.4).

Ces expérimentations n'ont donc pas permis de valider le modèle de puissance multi-cœurs, mais on mis en évidence un comportement du DVFS sur les processeurs "Intel I7" non soupçonné jusqu'ici.

Bibliographie

[1] Hady S Abdelsalam, Kurt Maly, Ravi Mukkamala, Mohammad Zubair et David Kaminsky. “Analysis of energy efficiency in clouds”. Dans : *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD'09. Computation World :* IEEE. 2009, p. 416–421 (cf. p. 40).

[2] B. Aksanli, J. Venkatesh et T.S. Rosing. “Using Datacenter Simulation to Evaluate Green Energy Integration”. Dans : *Computer* 45.9 (2012), p. 56–64 (cf. p. 49).

[3] M. Alexandru, T. Monteil, P. Lorenz, F. Coccetti et H. Aubert. “Efficient Large Electromagnetic Problem Solving by Hybrid TLM and Modal Approach on Grid Computing”. Dans : *International Microwave Symposium, Montréal, Canada, 17-22 june 2012*. 2012 (cf. p. 124).

[4] Phillip E Allen et Douglas R Holberg. *CMOS analog circuit design*. Oxford Univ. Press, 2002 (cf. p. 34).

[5] Cloud Security Alliance. *Security Guidance for Critical Areas of Focus in Cloud Computing V3.0*. `https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf` (cf. p. 73).

[6] *Amazon AWS*. `http://www.aws-ug.fr/` (cf. p. 5).

[7] *Amazon S3*. `http://aws.amazon.com/fr/s3/` (cf. p. 6).

[8] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica et Matei Zaharia. “A view of cloud computing”. Dans : *Commun. ACM* 53.4 (avr. 2010), p. 50–58 (cf. p. 3, 59).

[9] Remzi H. Arpaci-Dusseau et Andrea C. Arpaci-Dusseau. *Operating Systems : Three Easy Pieces*. Blackboard Books, 2012 (cf. p. 21).

[10] Daniel Ashlock. *Evolutionary computation for modeling and optimization*. T. 103. Springer, 2006 (cf. p. 31).

[11] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson et Dawn Song. “Provable data possession at untrusted stores”. Dans : *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007, p. 598–609 (cf. p. 70).

[12] Nader Azizi et Saeed Zolfaghari. “Adaptive Temperature Control for Simulated Annealing : A Comparative Study”. Dans : *Comput. Oper. Res.* 31.14 (déc. 2004), p. 2439–2451 (cf. p. 30).

[13] Slawomir Bak, Marcin Krystek, Krzysztof Kurowski, Ariel Oleksiak, Wojciech Piatek et Jan Waglarz. “GSSIM-A tool for distributed computing experiments.” Dans : *Scientific Programming* 19.4 (2011), p. 231–251 (cf. p. 47).

[14] J. Baliga, R.W.A. Ayre, K. Hinton et RodneyS. Tucker. “Green Cloud Computing : Balancing Energy in Processing, Storage, and Transport”. Dans : *Proceedings of the IEEE* 99.1 (2011), p. 149–167 (cf. p. 40).

[15] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt et Andrew Warfield. “Xen and the art of virtualization”. Dans : *ACM SIGOPS Operating Systems Review* 37.5 (2003), p. 164–177 (cf. p. 23).

[16] A. Beloglazov et R. Buyya. “Energy Efficient Resource Management in Virtualized Cloud Data Centers”. Dans : *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. 2010, p. 826–831 (cf. p. 41).

[17] Anton Beloglazov et Rajkumar Buyya. “Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers”. Dans : *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM. 2010, p. 4 (cf. p. 41).

[18] Micha vor dem Berge, Georges Da Costa, Mateusz Jarus, Ariel Oleksiak, Wojciech Piatek et Eugen Volk. “Modeling Data Center Building Blocks for Energy-Efficiency and Thermal Simulations”. Dans : *Energy-Efficient Data Centers*. Springer, 2014, p. 66–82 (cf. p. 48).

[19] Christian Blum. “Ant colony optimization : Introduction and recent trends”. Dans : *Physics of Life reviews* 2.4 (2005), p. 353–373 (cf. p. 31).

[20] Thomas D Burd et Robert W Brodersen. “Design issues for dynamic voltage scaling”. Dans : *Low Power Electronics and Design, 2000. ISLPED'00. Proceedings of the 2000 International Symposium on*. IEEE. 2000, p. 9–14 (cf. p. 13).

[21] Thomas D. Burd et Robert W. Brodersen. “Design issues for dynamic voltage scaling”. Dans : *Proceedings of the 2000 international symposium on Low power electronics and design*. ISLPED '00. New York, NY, USA : ACM, 2000, p. 9–14 (cf. p. 35).

[22] Rajkumar Buyya et Manzur Murshed. “GridSim : A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing”. Dans : *concurrency and computation : practice and experiene (ccpe)* 14.13 (2002), p. 1175–1220 (cf. p. 48).

[23] Juan Cáceres, Luis M Vaquero, Luis Rodero-Merino, Álvaro Polo et Juan J Hierro. “Service scalability over the cloud”. Dans : *Handbook of Cloud Computing*. Springer, 2010, p. 357–377 (cf. p. 64).

[24] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose et Rajkumar Buyya. “CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms”. Dans : *Software : Practice and Experience* 41.1 (2011), p. 23–50 (cf. p. 48, 102, 103).

[25] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier et O. Richard. “Grid'5000 : a large scale and highly reconfigurable grid experimental testbed”. Dans : *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*. 2005, 8 pp. (Cf. p. 109, 123).

[26] Henri Casanova, Arnaud Legrand et Martin Quinson. “SimGrid : A Generic Framework for Large-Scale Distributed Experiments”. Dans : *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*. 2008, p. 126–131 (cf. p. 47).

[27] Clovis Chapman, Wolfgang Emmerich, Fermın Galán Marquez, Stuart Clayman et Alex Galis. “Elastic service definition in computational clouds”. Dans : *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*. IEEE. 2010, p. 327–334 (cf. p. 5).

[28] I. Charon, A. Germa et O. Hudry. *Méthodes d'optimisation combinatoire*. Masson, 1996 (cf. p. 27).

[29] Weiwei Chen et Ewa Deelman. “Workflow overhead analysis and optimizations”. Dans : *Proceedings of the 6th workshop on Workflows in support of large-scale science*. WORKS '11. Seattle, Washington, USA : ACM, 2011, p. 11–20 (cf. p. 120).

[30] W. J. Choi, E. C C Yeh et K.N. Tu. “Mean-time-to-failure study of flip chip solder joints on Cu/Ni(V)/Al thin-film under-bump-metallization”. Dans : *Journal of Applied Physics* 94.9 (2003), p. 5665–5671. doi : 10.1063/1.1616993 (cf. p. 62).

[31] Lawrence Chung et Julio Cesar Prado Leite. “Conceptual Modeling : Foundations and Applications”. Dans : Berlin, Heidelberg : Springer-Verlag, 2009. Chap. On Non-Functional Requirements in Software Engineering, p. 363–379. doi : 10.1007/978-3-642-02463-4_19 (cf. p. 59).

[32] *Cloud9 analytics*. http://www.cloud9analytics.com/ (cf. p. 5).

[33] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo et al. "Distributed optimization by ant colonies". Dans : *Proceedings of the first European conference on artificial life*. T. 142. Paris, France. 1991, p. 134–142 (cf. p. 31).

[34] Intel Corporation. *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor White Paper*. 2004 (cf. p. 35).

[35] ILOG Cplex. "11.0 User's manual". Dans : *ILOG SA, Gentilly, France* (2007) (cf. p. 32, 161).

[36] G. Da Costa et H. Hlavacs. "Methodology of measurement for energy consumption of applications". Dans : *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*. IEEE. 2010, p. 290–297 (cf. p. 74).

[37] Yuan-Shun Dai, Bo Yang, Jack Dongarra et Gewei Zhang. "Cloud service reliability : Modeling and analysis". Dans : *The 15th IEEE Pacific Rim International Symposium on Dependable Computing*. 2009 (cf. p. 62).

[38] George B Dantzig, Alex Orden, Philip Wolfe et al. "The generalized simplex method for minimizing a linear form under linear inequality restraints". Dans : *Pacific Journal of Mathematics* 5.2 (1955), p. 183–195 (cf. p. 32).

[39] Gargi Dasgupta, Amit Sharma, Akshat Verma, Anindya Neogi et Ravi Kothari. "Workload Management for Power Efficiency in Virtualized Data Centers". Dans : *Commun. ACM* 54.7 (juil. 2011), p. 131–141 (cf. p. 41).

[40] Corentin Dupont, Giovanni Giuliani, Fabien Hermenier, Thomas Schulze et Andrey Somov. "An energy aware framework for virtual machine placement in cloud federated data centres". Dans : *Future Energy Systems : Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*. IEEE. 2012, p. 1–10 (cf. p. 13).

[41] Truong Vinh Truong Duy, Yukinori Sato et Yasushi Inoguchi. "Performance evaluation of a green scheduling algorithm for energy savings in cloud computing". Dans : *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE. 2010, p. 1–8 (cf. p. 41).

[42] I.P. Egwutuoha, Shiping Chen, D. Levy, B. Selic et R. Calvo. "Energy Efficient Fault Tolerance for High Performance Computing (HPC) in the Cloud". Dans : *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. 2013, p. 762–769 (cf. p. 67).

[43] Dag Elgesem. "The structure of rights in Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and the free movement of such data". Dans : *Ethics and Information Technology* 1.4 (1999), p. 283–293 (cf. p. 70).

[44] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou et Roberto Tamassia. "Dynamic provable data possession". Dans : *Proceedings of the 16th ACM conference on Computer and communications security*. ACM. 2009, p. 213–222 (cf. p. 70).

[45] Dror Feitelson. "Parallel workloads archive". Dans : 71.86 (2007), p. 337–360 (cf. p. 48).

[46] Krisztián Flautner, Steve Reinhardt et Trevor Mudge. "Automatic performance setting for dynamic voltage scaling". Dans : *Proceedings of the 7th annual international conference on Mobile computing and networking*. ACM. 2001, p. 260–271 (cf. p. 36).

[47] Gérard Fleury. "Méthodes stochastiques et déterministes pour les problèmes NP-difficiles". Thèse de doct. 1993 (cf. p. 29).

[48] Carlos M Fonseca et Peter J Fleming. "An overview of evolutionary algorithms in multiobjective optimization". Dans : *Evolutionary computation* 3.1 (1995), p. 1–16 (cf. p. 30).

[49] Ian Foster, Yong Zhao, Ioan Raicu et Shiyong Lu. "Cloud computing and grid computing 360-degree compared". Dans : *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee. 2008, p. 1–10 (cf. p. 3).

[50] A. S. Fraser. "Simulation of genetic systems by automatic digital computers. I. Introduction". Dans : *Australian Journal of Biological Science* 10 (1957), p. 484–491 (cf. p. 30).

[51] Michael R Garey et David S Johnson. "Computer and intractability". Dans : *A Guide to the NP-Completeness. Ney York, NY : WH Freeman and Company* (1979) (cf. p. 14, 79).

[52] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam et Rajkumar Buyya. “Environment-conscious scheduling of {HPC} applications on distributed Cloud-oriented data centers”. Dans : *Journal of Parallel and Distributed Computing,Special Issue on Cloud Computing* 71.6 (2011), p. 732 –749 (cf. p. 40).

[53] DP Gaver. “Time to failure and availability of paralleled systems with repair”. Dans : *Reliability, IEEE Transactions on* 12.2 (1963), p. 30–38 (cf. p. 62).

[54] Erol Gelenbe, Ricardo Lent et Markos Douratsos. “Choosing a local or remote cloud”. Dans : *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*. IEEE. 2012, p. 25–30 (cf. p. 40).

[55] E. Ghazizadeh, J.-L.A. Manan, M. Zamani et A. Pashang. “A survey on security issues of federated identity in the cloud computing”. Dans : *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. 2012, p. 532–565. doi : 10.1109/CloudCom.2012.6427513 (cf. p. 70).

[56] *Google App Engine*. https://appengine.google.com/ (cf. p. 3).

[57] *Google Compute Engine*. https://cloud.google.com/products/compute-engine/ (cf. p. 5).

[58] Kinshuk Govil, Edwin Chan et Hal Wasserman. “Comparing algorithm for dynamic speed-setting of a low-power CPU”. Dans : *Proceedings of the 1st annual international conference on Mobile computing and networking*. ACM. 1995, p. 13–25 (cf. p. 36).

[59] Pankaj Goyal. “Enterprise usability of cloud computing environments : issues and challenges”. Dans : *Enabling Technologies : Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*. IEEE. 2010, p. 54–59 (cf. p. 69).

[60] “Grid’5000 : A Large Scale And Highly Reconfigurable Experimental Grid Testbed”. Dans : *International Journal of High Performance Computing Applications* 20.4 (nov. 2006), p. 481–494. doi : 10.1177/1094342006070078 (cf. p. 109, 123).

[61] Tom Guérout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya et Mihai Alexandru. “Energy-aware simulation with DVFS”. Dans : *Simulation Modelling Practice and Theory* 39 (2013), p. 76–91 (cf. p. 17, 74).

[62] Sebastian Herbert et Diana Marculescu. “Analysis of dynamic voltage/frequency scaling in chip-multiprocessors”. Dans : *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*. IEEE. 2007, p. 38–43 (cf. p. 34).

[63] W.J.R. Hoeffer. “The Transmission-Line Matrix Method–Theory and Applications”. Dans : *Microwave Theory and Techniques, IEEE Transactions on* 33.10 (1985), p. 882–893 (cf. p. 123).

[64] *HP Enterprise Converged Infrastructure*. http://h17007.www1.hp.com/fr/fr/converged-infrastructure/ (cf. p. 5).

[65] *IBM SmartCloud Enterprise*. http://www.ibm.com/cloud-computing/fr/fr/iaas.html (cf. p. 5).

[66] Tohru Ishihara et Hiroto Yasuura. “Voltage scheduling problem for dynamically variable voltage processors”. Dans : *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*. IEEE. 1998, p. 197–202 (cf. p. 35).

[67] Sadeka Islam, Kevin Lee, Alan Fekete et Anna Liu. “How a consumer can measure elasticity for cloud platforms”. Dans : *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ICPE ’12. Boston, Massachusetts, USA : ACM, 2012, p. 85–96. doi : 10.1145/2188286.2188301 (cf. p. 40).

[68] M. Jensen, J. Schwenk, N. Gruschka et L.L. Iacono. “On Technical Security Issues in Cloud Computing”. Dans : *Cloud Computing, 2009. CLOUD ’09. IEEE International Conference on*. 2009, p. 109–116. doi : 10.1109/CLOUD.2009.60 (cf. p. 70).

[69] Ari Juels et Burton S Kaliski Jr. “PORs : Proofs of retrievability for large files”. Dans : *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007, p. 584–597 (cf. p. 70).

[70] John D Kalbfleisch et Ross L Prentice. *The statistical analysis of failure time data*. T. 360. John Wiley & Sons, 2011 (cf. p. 62).

[71] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari et Arka A Bhattacharya. “Virtual machine power metering and provisioning”. Dans : *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. 2010, p. 39–50 (cf. p. 34).

[72] Christopher Clark Keir, Christopher Clark, Keir Fraser, Steven H, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt et Andrew Warfield. “Live Migration of Virtual Machines”. Dans : *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2005, p. 273–286 (cf. p. 33).

[73] H. Kimura, M. Sato, Y. Hotta, T. Boku et D. Takahashi. “Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster”. Dans : *Cluster Computing, 2006 IEEE International Conference on*. IEEE. 2006, p. 1–10 (cf. p. 104, 120).

[74] S. Kirkpatrick, C. D. Gelatt et M. P. Vecchi. “Optimization by simulated annealing”. Dans : *SCIENCE* 220.4598 (1983), p. 671–680 (cf. p. 29).

[75] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin et Anthony Liguori. “kvm : the Linux virtual machine monitor”. Dans : *Proceedings of the Linux Symposium*. T. 1. 2007, p. 225–230 (cf. p. 26).

[76] D. Kliazovich, P. Bouvry, Y. Audzevich et S.U. Khan. “GreenCloud : A Packet-Level Simulator of Energy-Aware Cloud Computing Data Centers”. Dans : *GLOBECOM 2010, IEEE Global Telecommunications Conference*. 2010, p. 1–5 (cf. p. 45).

[77] R.K.L. Ko, P. Jagadpramana, M. Mowbray, S. Pearson, M. Kirchberg, Qianhui Liang et Bu Sung Lee. “TrustCloud : A Framework for Accountability and Trust in Cloud Computing”. Dans : *Services (SERVICES), 2011 IEEE World Congress on*. 2011, p. 584–588. doi : 10.1109/SERVICES.2011.91 (cf. p. 71).

[78] M. Kocaoglu, D. Malak et O.B. Akan. “Fundamentals of Green Communications and Computing : Modeling and Simulation”. Dans : *Computer* 45.9 (2012), p. 40–46 (cf. p. 49).

[79] T. Kolpe, A. Zhai et S.S. Sapatnekar. “Enabling improved power management in multicore processors through clustered DVFS”. Dans : *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*. 2011, p. 1 –6 (cf. p. 34).

[80] Tejaswini Kolpe, Antonia Zhai et Sachin S Sapatnekar. “Enabling improved power management in multicore processors through clustered DVFS”. Dans : *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE. 2011, p. 1–6 (cf. p. 104).

[81] Jonathan Koomey. “Growth in data center electricity use 2005 to 2010”. Dans : *A report by Analytical Press, completed at the request of The New York Times* (2011) (cf. p. 9).

[82] K. Kritikos, B. Pernici, P. Plebani, C. Cappiello, M. Comuzzi, S. Benbernou, I. Brandic, A. Kertesz, M. Parkin et M. Caro. “A Survey on Service Quality Description.” Dans : *ACM Transactions Computing Survey* (July 2012) (cf. p. 59).

[83] K. Kurowski, J. Nabrzyski, A. Oleksiak et J. Weglarz. “Grid scheduling simulations with GSSIM”. Dans : *Parallel and Distributed Systems, 2007 International Conference on*. T. 2. 2007, p. 1–8 (cf. p. 47).

[84] Krzysztof Kurowski, Ariel Oleksiak, W Piatek, Tomasz Piontek, A Przybyszewski et J Weglarz. “DCworms–A tool for simulation of energy efficiency in distributed computing infrastructures”. Dans : *Simulation Modelling Practice and Theory* 39 (2013), p. 135–151 (cf. p. 47).

[85] Daniel Guimaraes do Lago, Edmundo R. M. Madeira et Luiz Fernando Bittencourt. “Power-aware Virtual Machine Scheduling on Clouds Using Active Cooling Control and DVFS”. Dans : *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*. MGC ’11. New York, NY, USA : ACM, 2011, 2 :1–2 :6 (cf. p. 133).

[86] G. von Laszewski, Lizhe Wang, AJ. Younge et Xi He. “Power-aware scheduling of virtual machines in DVFS-enabled clusters”. Dans : *Cluster Computing and Workshops, 2009. CLUSTER ’09. IEEE International Conference on*. 2009, p. 1–10 (cf. p. 104).

[87] Jan Karel Lenstra, AHG Rinnooy Kan et Peter Brucker. “Complexity of machine scheduling problems”. Dans : *Annals of discrete mathematics* 1 (1977), p. 343–362 (cf. p. 79).

[88] Jinyuan Li, Maxwell N Krohn, David Mazières et Dennis Shasha. “Secure untrusted data repository (SUNDR)”. Dans : *OSDI*. T. 4. 2004, p. 9–9 (cf. p. 71).

[89] Haikun Liu, Hai Jin, Cheng-Zhong Xu et Xiaofei Liao. “Performance and energy modeling for live migration of virtual machines”. Dans : *Cluster computing* 16.2 (2013), p. 249–264 (cf. p. 34).

[90] Yutu Liu, Anne H Ngu et Liang Z Zeng. "QoS computation and policing in dynamic web service selection". Dans : *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM. 2004, p. 66–73 (cf. p. 59).

[91] M Lori. "Data security in the world of cloud computing". Dans : *Co-published by the IEEE Computer And reliability Societies* (2009), p. 61–64 (cf. p. 70).

[92] Zaigham Mahmood. "Cloud computing : Characteristics and deployment approaches". Dans : *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*. IEEE. 2011, p. 121–126 (cf. p. 6).

[93] Tim Mather, Subra Kumaraswamy et Shahed Latif. *Cloud security and privacy : an enterprise perspective on risks and compliance*. O'Reilly, 2009 (cf. p. 70).

[94] Peter Mell et Timothy Grance. "The NIST definition of cloud computing (draft)". Dans : *NIST special publication* 800.145 (2011), p. 7 (cf. p. 3, 59).

[95] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". Dans : *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*. CRYPTO '87. London, UK, UK : Springer-Verlag, 1988, p. 369–378 (cf. p. 71).

[96] Haibo Mi, Huaimin Wang, Gang Yin, Yangfan Zhou, Dianxi Shi et Lin Yuan. "Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers". Dans : *Services Computing (SCC), 2010 IEEE International Conference on*. IEEE. 2010, p. 514–521 (cf. p. 41).

[97] *Microsoft Azure*. `http://azure.microsoft.com/fr-fr/` (cf. p. 5).

[98] S. Nakahara et H. Ishimoto. "A study on the requirements of accountable cloud services and log management". Dans : *Information and Telecommunication Technologies (APSITT), 2010 8th Asia-Pacific Symposium on*. 2010, p. 1–6 (cf. p. 72).

[99] Roger M Needham et Michael D Schroeder. "Using encryption for authentication in large networks of computers". Dans : *Communications of the ACM* 21.12 (1978), p. 993–999 (cf. p. 73).

[100] F. Nerieri, R. Prodan, T. Fahringer et H.L. Truong. "Overhead analysis of grid workflow applications". Dans : *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*. IEEE Computer Society. 2006, p. 17–24 (cf. p. 120).

[101] Fengming Nie, Feng Xu et Rongzhi Qi. "SAML-based single sign-on for legacy system". Dans : *Automation and Logistics (ICAL), 2012 IEEE International Conference on*. 2012, p. 470–473. doi : 10.1109/ICAL.2012.6308228 (cf. p. 70).

[102] Simon Ostermann, Kassian Plankensteiner, Radu Prodan et Thomas Fahringer. "GroudSim : An Event-based Simulation Framework for Computational Grids and Clouds". Dans : *Euro-Par 2010 – Parallel Processing Workshops*. Lecture Notes in Computer Science. Springer, 2011 (cf. p. 45).

[103] Jitendra Padhye, Victor Firoiu, Don Towsley et Jim Kurose. "Modeling TCP throughput : a simple model and its empirical validation". Dans : *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*. SIGCOMM '98. Vancouver, British Columbia, Canada : ACM, 1998, p. 303–314. doi : 10.1145/285237.285291 (cf. p. 60).

[104] An Oracle White paper. *Information Lifecycle Management for Business Data*. `http://www.oracle.com/us/026964.pdf`. 2007 (cf. p. 73).

[105] M.K. Patterson. "Energy Efficiency Metrics". Dans : *Energy Efficient Thermal Management of Data Centers, Springer, 2012* (2012) (cf. p. 69).

[106] W.A. Pauley. "Cloud Provider Transparency : An Empirical Evaluation". Dans : *Security Privacy, IEEE* 8.6 (2010), p. 32–39. doi : 10.1109/MSP.2010.140 (cf. p. 59).

[107] Michael G Pecht et FR Nash. "Predicting the reliability of electronic equipment [and prolog]". Dans : *Proceedings of the IEEE* 82.7 (1994), p. 992–1004 (cf. p. 62).

[108] Trevor Pering, Tom Burd et Robert Brodersen. "The simulation and evaluation of dynamic voltage scaling algorithms". Dans : *Proceedings of the 1998 international symposium on Low power electronics and design*. ACM. 1998, p. 76–81 (cf. p. 34).

[109] Padmanabhan Pillai et Kang G Shin. “Real-time dynamic voltage scaling for low-power embedded operating systems”. Dans : *ACM SIGOPS Operating Systems Review*. T. 35. 5. ACM. 2001, p. 89–102 (cf. p. 36).

[110] Krishna PN Puttaswamy, Christopher Kruegel et Ben Y Zhao. “Silverline : toward data confidentiality in storage-intensive cloud applications”. Dans : *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM. 2011, p. 10 (cf. p. 70, 71).

[111] *Rackspace Cloud*. http://www.rackspace.com/cloud/http ://www.rackspace.com/cloud/ (cf. p. 5).

[112] Vincent Rainardi. “Functional and Nonfunctional Requirements”. Dans : *Building a Data Warehouse : With Examples in SQL Server* (2008), p. 61–70 (cf. p. 59).

[113] T Rajendran, P Balasubramanie et Resmi Cherian. “An efficient WS-QoS broker based architecture for web services selection”. Dans : *International Journal of Computer Applications* 1.9 (2010), p. 110–115 (cf. p. 59).

[114] Shuping Ran. “A model for web services discovery with QoS”. Dans : *ACM Sigecom exchanges* 4.1 (2003), p. 1–10 (cf. p. 59).

[115] Bhaskar Prasad Rimal, Eunmi Choi et Ian Lumb. “A taxonomy and survey of cloud computing systems”. Dans : *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*. Ieee. 2009, p. 44–51 (cf. p. 3, 59).

[116] Suzanne Rivoire, Parthasarathy Ranganathan et Christos Kozyrakis. “A comparison of high-level full-system power models”. Dans : *Proceedings of the 2008 conference on Power aware computing and systems*. HotPower'08. San Diego, California : USENIX Association, 2008, p. 3–3 (cf. p. 74).

[117] M. Rosenblum et T. Garfinkel. “Virtual machine monitors : current technology and future trends”. Dans : *Computer* 38.5 (2005), p. 39–47 (cf. p. 21).

[118] Rusty Russell. “virtio : towards a de-facto standard for virtual I/O devices”. Dans : *ACM SIGOPS Operating Systems Review* 42.5 (2008), p. 95–103 (cf. p. 25).

[119] K. Rybina, W. Dargie, A. Strunk et A. Schill. “Investigation into the energy cost of live migration of virtual machines”. Dans : *Sustainable Internet and ICT for Sustainability (SustainIT), 2013*. 2013, p. 1–8 (cf. p. 34).

[120] Ahmad-Reza Sadeghi, Thomas Schneider et Marcel Winandy. “Token-Based Cloud Computing”. Dans : *Trust and Trustworthy Computing*. Sous la dir. d'Alessandro Acquisti, SeanW. Smith et Ahmad-Reza Sadeghi. T. 6101. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, p. 417–429. doi : 10.1007/978-3-642-13869-0_30 (cf. p. 70).

[121] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein et Charles E Youman. “Role-based access control models”. Dans : *Computer* 29.2 (1996), p. 38–47 (cf. p. 70).

[122] Ichiro Satoh. “A Scheme for Carbon Offsetting Products and Services”. Dans : *Commerce and Enterprise Computing (CEC), 2011 IEEE 13th Conference on*. IEEE. 2011, p. 201–206 (cf. p. 75).

[123] Steve Versteeg Saurabh Kumar Garg et Rajkumar Buyya. “SMICloud : A Framework for Comparing and Ranking Cloud Services”. Dans : *Proceedings of the 4th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2011, IEEE CS Press, USA), Melbourne, Australia*. 2011 (cf. p. 59).

[124] Laura Savu. “Cloud computing : Deployment models, delivery models, risks and research challenges”. Dans : *Computer and Management (CAMAN), 2011 International Conference on*. IEEE. 2011, p. 1–4 (cf. p. 6).

[125] B. Schroeder et G.A. Gibson. “A Large-Scale Study of Failures in High-Performance Computing Systems”. Dans : *Dependable and Secure Computing, IEEE Transactions on* 7.4 (2010), p. 337–350 (cf. p. 63).

[126] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu et John Wilkes. “CloudScale : Elastic Resource Scaling for Multi-tenant Cloud Systems”. Dans : *Proceedings of the 2Nd ACM Symposium on Cloud Computing*. SOCC '11. Cascais, Portugal : ACM, 2011, 5 :1–5 :14. doi : 10.1145/2038916.2038921 (cf. p. 104).

[127] Vijayaraghavan Soundararajan et Kinshuk Govil. “Challenges in Building Scalable Virtualized Datacenter Management”. Dans : *SIGOPS Oper. Syst. Rev.* 44.4 (déc. 2010), p. 95–102. doi : 10.1145/1899928.1899941 (cf. p. 3).

[128] William M Spears, Kenneth A De Jong, Thomas Bäck, David B Fogel et Hugo De Garis. “An overview of evolutionary computation”. Dans : *Machine Learning : ECML-93*. Springer. 1993, p. 442–459 (cf. p. 30).

[129] Shekhar Srikantaiah, Aman Kansal et Feng Zhao. "Energy Aware Consolidation for Cloud Computing". Dans : *Proceedings of the 2008 Conference on Power Aware Computing and Systems*. HotPower'08. Berkeley, CA, USA : USENIX Association, 2008, p. 10–10 (cf. p. 41).

[130] M. Srinivas et L.M. Patnaik. "Genetic algorithms : a survey". Dans : *Computer* 27.6 (1994), p. 17–26 (cf. p. 83).

[131] Mark Stillwell, David Schanzenbach, Frédéric Vivien et Henri Casanova. "Resource allocation algorithms for virtualized service hosting platforms". Dans : *Journal of Parallel and Distributed Computing* 70.9 (2010), p. 962–974 (cf. p. 53).

[132] A. Strunk. "Costs of Virtual Machine Live Migration : A Survey". Dans : *Services (SERVICES), 2012 IEEE Eighth World Congress on*. 2012, p. 323–329 (cf. p. 34).

[133] Anja Strunk. "A Lightweight Model for Estimating Energy Cost of Live Migration of Virtual Machines". Dans : *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE. 2013, p. 510–517 (cf. p. 34).

[134] S. Subashini et V. Kavitha. "A survey on security issues in service delivery models of cloud computing". Dans : *Journal of Network and Computer Applications* 34.1 (2011), p. 1 –11. doi : http://dx.doi.org/10.1016/j.jnca.2010.07.006 (cf. p. 70).

[135] Jeremy Sugerman, Ganesh Venkitachalam et Beng-Hong Lim. "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor." Dans : *USENIX Annual Technical Conference, General Track*. 2001, p. 1–14 (cf. p. 22).

[136] Haluk Topcuoglu, Salim Hariri et Min-you Wu. "Performance-effective and low-complexity task scheduling for heterogeneous computing". Dans : *Parallel and Distributed Systems, IEEE Transactions on* 13.3 (2002), p. 260–274 (cf. p. 79).

[137] Hélène Toussaint. "Algorithmique rapide pour les problèmes de tournées et d'ordonnancement". Thèse de doct. Université Blaise Pascal-Clermont-Ferrand II, 2010 (cf. p. 26).

[138] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres et Maik Lindner. "A break in the clouds : towards a cloud definition". Dans : *ACM SIGCOMM Computer Communication Review* 39.1 (2008), p. 50–55 (cf. p. 5).

[139] A. Vasan et Komaragiri Srinivasa Raju. "Comparative Analysis of Simulated Annealing, Simulated Quenching and Genetic Algorithms for Optimal Reservoir Operation". Dans : *Appl. Soft Comput.* 9.1 (jan. 2009), p. 274–281 (cf. p. 29).

[140] V. Černý. "Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm". Dans : *Journal of Optimization Theory and Applications* 45.1 (jan. 1985), p. 41–51 (cf. p. 29).

[141] Lizhe Wang, Gregor von Laszewski, Jay Dayal et Fugang Wang. "Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS". Dans : *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. CCGRID '10. Washington, DC, USA : IEEE Computer Society, 2010, p. 368–377. doi : 10.1109/CCGRID.2010.19 (cf. p. 104).

[142] Michael W Wara et David G Victor. *A realistic policy on international carbon offsets*. Rap. tech. Working paper, 2008 (cf. p. 75).

[143] Mark Weiser, Brent Welch, Alan Demers et Scott Shenker. "Scheduling for reduced CPU energy". Dans : *Mobile Computing*. Springer, 1996, p. 449–471 (cf. p. 36).

[144] J. White III et S. Bova. "Where's the overlap ? An analysis of popular MPI implementations". Dans : *Proceedings of the Third MPI Developers' and Users' Conference*. Citeseer. 1999 (cf. p. 123).

[145] Thomas Wiedmann et Jan Minx. "A definition of'carbon footprint'". Dans : *Ecological economics research trends* 2 (2007), p. 55–65 (cf. p. 75).

[146] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat et Per Stenström. "The worst case execution time problem overview of methods and survey of tools". Dans : *ACM Trans. Embed. Comput. Syst.* 7.3 (mai 2008), 36 :1–36 :53. doi : 10.1145/1347375.1347389 (cf. p. 60).

[147] Wei Wu, Jianying Zhou, Yang Xiang et Li Xu. “How to achieve non-repudiation of origin with privacy protection in cloud computing”. Dans : *Journal of Computer and System Sciences* 79.8 (2013), p. 1200–1213 (cf. p. 74).

[148] Zhifeng Xiao et Yang Xiao. “Security and Privacy in Cloud Computing”. Dans : *Communications Surveys Tutorials, IEEE* 15.2 (2013), p. 843–859. doi : 10.1109/SURV.2012.060912.00182 (cf. p. 72).

[149] Shixing Yan, Bu Sung Lee, Guopeng Zhao, Ding Ma et Peer Mohamed. “Infrastructure management of hybrid cloud for enterprise users”. Dans : *Systems and Virtualization Management (SVM), 2011 5th International DMTF Academic Alliance Workshop on*. IEEE. 2011, p. 1–6 (cf. p. 6).

[150] Frances Yao, Alan Demers et Scott Shenker. “A scheduling model for reduced CPU energy”. Dans : *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*. IEEE. 1995, p. 374–382 (cf. p. 36).

[151] AJ. Younge, R. Henschel, J.T. Brown, G. von Laszewski, J. Qiu et G.C. Fox. “Analysis of Virtualization Technologies for High Performance Computing Environments”. Dans : *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. 2011, p. 9–16 (cf. p. 26).

[152] Lamia Youseff, Maria Butrico et Dilma Da Silva. “Toward a unified ontology of cloud computing”. Dans : *Grid Computing Environments Workshop, 2008. GCE’08*. IEEE. 2008, p. 1–10 (cf. p. 3).

[153] Alexander Zahariev. “TKK T-110.5190 Seminar on Internetworking”. Dans : *Google App Engine*. Helsinki University of Technolog, 2009 (cf. p. 3, 5).

[154] Gansen Zhao, Chunming Rong, Martin Gilje Jaatun et Frode Eika Sandnes. “Deployment models : Towards eliminating security concerns from cloud computing”. Dans : *High Performance Computing and Simulation (HPCS), 2010 International Conference on*. IEEE. 2010, p. 189–195 (cf. p. 6).

[155] Jiantao Zhou, Shang Zheng, Delin Jing et Hongji Yang. “An approach of creative application evolution on cloud computing platform”. Dans : *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM. 2011, p. 54–58 (cf. p. 5).