Gestion de ressources de façon "éco-énergétique" dans un système virtualisé : application à l'ordonnanceur de marchines virtuelles

Christine Mayap Kamga

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Réseaux, Télécommunications, Systèmes et Architecture

Présentée et soutenue par :

Mme CHRISTINE MAYAP KAMGA

le jeudi 26 juin 2014

Titre :

GESTION DE RESSOURCES DE FACON "ECO-ENERGETIQUE" DANS UN SYSTEME VIRTUALISE: APPLICATION A L'ORDONNANCEUR DE MARCHINES VIRTUELLES

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (I.R.I.T.)

Directeur(s) de Thèse :

M. DANIEL HAGIMONT

Rapporteurs :

M. GILLES GRIMAUD, UNIVERSITE LILLE 1

Mme FRANCOISE BAUDE, INRIA SOPHIA ANTIPOLIS

Membre(s) du jury :

M. NOËL DE PALMA, UNIVERSITE GRENOBLE 1, Président

M. ALAIN TCHANA, INP TOULOUSE, Membre

M. DANIEL HAGIMONT, INP TOULOUSE, Membre

M. FABIEN HERMENIER, INRIA SOPHIA ANTIPOLIS, Membre

M. JEAN-MARC PIERSON, UNIVERSITE TOULOUSE 3, Membre

A ma famille pour leur soutien
A Désiré pour son amour et sa présence

"Those who sow with tears will reap with songs of joy. Those who go out weeping, carrying seed to sow, will return with songs of joy, carrying sheaves with them." .
Psalms.

Je tiens tout d'abord à remercier tous les membres du jury pour le temps que vous avez consacré à l'évaluation de mes travaux de recherche. Merci aux rapporteurs Françoise BAUDE, Fabien HERMENIER et Gilles GRIMAUD pour avoir pris soin de rapporter ma thèse. Je tiens à les remercier pour leur évaluation critique sur le fond et la forme. Merci pour les remarques constructives et les interrogations qui m'ont permis de considérer d'autres aspects pour des travaux futurs. Merci à Noel DE PALMA pour avoir accepté de présider ce jury de soutenance.

J'exprime ma profonde reconnaissance à mon directeur de thèse Daniel HAGIMONT pour son encadrement durant toutes ces années de thèse. Merci pour tous ses conseils avisés, sa pédagogie et le temps qu'il m'a consacré pour que je puisse mener cette thèse à son terme. Merci pour sa rigueur lors des expériences qui m'ont valu de produire des articles de qualité. Merci pour sa quête de précision et de concision qui ont développé en moi de meilleures aptitudes rédactionnelles, appréciées par mes rapporteurs. Je tiens à remercier Laurent BROTO pour m'avoir offert l'opportunité de réaliser cette thèse. Merci pour ton accueil dans l'équipe, l'aide à l'intégration dans ce nouveau pays et surtout pour ton expertise technique.

Un merci spécial à nos adorables secrétaires Sylvie ARMENGAUD et Sylvie EICHEN. Vous êtes exceptionnelles dans votre travail et vous avez contribué à faciliter le mien. Merci pour la bonne humeur, pour le petit mail de rappel pour ceux avait souvent une petite faim :)

Merci à tous les membres de l'équipe SEPIA pour leur accueil. Je remercie plus particulièrement tous les docteurs/doctorants du site IRIT/ENSEEIHT (Suzy, Aeiman, Alain, Son, Esso, Momo et Brice) pour les moments très agréables et toutes les blagues que nous avons partagées. Vos encouragements et la détermination dans vos travaux respectifs m'ont aidé à persévérer jusqu'au bout. Je remercie tous les autres docteurs (Bafing, Omer, Bagayoko) avec qui j'ai eu à partager de nombreux repas (français, camerounais, malien ou sénégalais) ainsi que les joies et les difficultés de la vie de doctorant.

Merci à tous mes ami(e)s en dehors du cadre professionnel (Linda, Cindy, Christiane, Florentine, Edwige&Thibaut, Pauline, Anne-Estelle, Arlete&Claude, Brigitte, Ingrid, Nadège, Léonie, les couples Duret, Dzeukou, Souopgui, Donfack, Chakode, Tchoupé, Mangoua, Tchuenté, Deuheula, Leutchou, Douba,Kouamen et les autres qui m'excuseront de ne pas les nommer). Merci à ma famille chrétienne (OLIC, GBU, 3Jean2, GBEEC, UCJG) ainsi qu'à tous mes amis du Cameroun. J'ai pu apprécier votre amitié durant ces années et j'ai toujours pu compter sur votre soutien.

Enfin, je garde une attention toute particulière à toute ma grande famille (les familles Kamguem, Ndze, Foné, Nono, Lemegne, Heutchou, Tamba, Fotso, Wabo,

Tamdjo, Ngansop), pour tout ce qu'elle a pu (et continue à) m'apporter. Merci à mes parents pour m'avoir donné ce qu'ils avaient de meilleur pour assurer la réussite de mes études. Merci de m'avoir apporté, en plus de votre amour, le goût de la réussite et du rêve. Merci à mon père (*Mr KAMGA, j'aime prononcer ce nom... :)*) qui a toujours su m'encourager à sa façon. Merci à ma maman chérie, qui à travers sa joie de vivre me tenait au parfum des évènements survenant dans ma famille restée au Cameroun. J'espère vous avoir honoré en achevant avec brio ce projet de thèse. Merci à mes frères et sœurs pour votre présence...vous m'êtes précieux. J'espère vous avoir appris une leçon de quelques ordres que ce soit.

Un merci très très spécial à Désiré Nuentsa : une rencontre, des échanges, des blagues, des encouragements, un amour....Ta présence a grandement contribué à l'aboutissement de cette thèse. Tu as su m'encourager, me rappeler les priorités, me détendre et me faire profiter de ton expérience. Merci d'avoir été là et de continuer à être là.

Face au coût de la gestion locale des infrastructures informatiques, de nombreuses entreprises ont décidé de la faire gérer par des fournisseurs externes. Ces derniers, connus sous le nom de IaaS (Infrastructure as a Service), mettent des ressources à la disposition des entreprises sous forme de machines virtuelles (VM - Virtual Machine). Ainsi, les entreprises n'utilisent qu'un nombre limité de machines virtuelles capables de satisfaire leur besoin. Ce qui contribue à la réduction des coûts de l'infrastructure informatique des entreprises clientes. Cependant, cette externalisation soulève pour le fournisseur, les problèmes de respect d'accord de niveau de service (SLA - Service Layer Agreement) souscrit par le client et d'optimisation de la consommation énergétique de son infrastructure.

Au regard de l'importance que revêt ces deux défis, de nombreux travaux de recherche se sont intéressés à cette problématique. Les solutions de gestion d'énergie proposées consistent à faire varier la vitesse d'exécution des périphériques concernés. Cette variation de vitesse est implémentée, soit de façon native parce que le périphérique dispose des mécaniques intégrés, soit par simulation à travers des regroupements (spatial et temporel) des traitements. Toutefois, cette variation de vitesse permet d'optimiser la consommation énergétique d'un périphérique mais, a pour effet de bord d'impacter le niveau de service des clients. Cette situation entraine une incompatibilité entre les politiques de variation de vitesse utilisées pour la baisse d'énergie et le respect de l'accord de niveau de service.

Dans cette thèse, nous étudions la conception et l'implantation d'un gestionnaire de ressources "*éco-énergétique*" dans un système virtualisé. Un tel gestionnaire doit permettre un partage équitable des ressources entre les machines virtuelles tout en assurant une utilisation optimale de l'énergie que consomment ces ressources. Nous illustrons notre étude avec un ordonnanceur de machines virtuelles. La politique de variation de vitesse est implantée par le DVFS (Dynamic Voltage Frequency Scaling) et l'allocation de la capacité CPU aux machines virtuelles représente l'accord de niveau de service à respecter.

Mots-clés : Ordonnanceur, SLA, Machine virtuelle, Energie, Reconfiguration, Hyperviseur

Considering the cost of local management of the computing infrastructures, numerous companies decided to delegate theirs to providers. These latter are known as an Infrastructure as a Service (IaaS) and provide resources to companies in the form of virtual machine (VM). This decision to outsource contributes to lower the cost of IT infrastructure of the customer companies. However, it raises for the provider, the problems of the respect of the Service Layer agreement (SLA) of the customer and of the optimization of the energy consumption of his infrastructure

With regard to the importance of these two challenges, many research works have focused on this problem. The proposed energy management solutions consist in varying the execution speed of the affected devices. This variation of speed is implemented either natively because the device has integrated mechanics, or by simulation through a spatial or temporal batching requests. However, this variation of speed optimizes the energy consumption of a device but has the side effect of degrading the customers SLA.

In this thesis, we study the design and the implementation of an energy-efficient resources manager in a virtualized system. Such a manager must ensures a fair-share of resources among VMs while ensuring optimal use of the energy consumed by the resources. We illustrate our study thanks to a scheduler of VMs. The DVFS constitutes our energy management policy and the CPU capacity of the VMs the SLA to respect.

Table des matières

Chapitre 1

Introduction

Au cours de ces dernières années, de nombreuses entreprises (petite et moyenne) ont vu s'accroître le coût de fonctionnement (mise en place et maintenance) de leur infrastructure informatique. L'augmentation de ce coût (matériel, logiciel, humain et énergétique) est fonction du nombre d'équipements que contient cette infrastructure d'une part ; et du choix de déploiement (redondance des services) adopté d'autre part. Face à cette croissance, plusieurs entreprises ont entrepris d'externaliser la gestion de leur infrastructure auprès des fournisseurs de services.
Ces derniers se caractérisent par le modèle de service qu'ils offrent : machine virtuelle dans le cas d'un IaaS, plateforme logicielle dans le cas d'un PaaS et logiciels dans le cas d'un SaaS. Quel que soit le service offert, ces fournisseurs rencontrent de nombreux problèmes notamment : la sécurité et la confidentialité des données des clients, le respect de l'accord de niveau de service (SLA - Service Layer Agreement) souscrit par le client et la réduction de la consommation énergétique de leur infrastructure. Dans le cadre de cette thèse, nous nous focalisons sur le respect de l'accord de niveau de service et la baisse de la consommation énergétique.

Le respect de l'accord de niveau de service et la gestion de la consommation d'énergie nécessitent d'adopter des politiques de gestion dynamique d'énergie. De nombreuses approches ont ainsi été proposées. Sur la base d'une étude que nous avons effectuée, nous avons observé que l'ensemble de ces politiques sont, en général, basées sur la variation de la vitesse d'exécution du périphérique sur lequel la politique de gestion d'énergie est appliquée. Toutefois, cette variation a comme inconvénient de réduire la vitesse d'accès/d'utilisation de la ressource concernée. Ce qui entraine le non respect du niveau de service des machines virtuelles qui l'utilisent. Ainsi, afin de garantir l'accord de niveau de service de ses clients et réduire la consommation énergétique de son infrastructure, le fournisseur de service doit définir et mettre en place des stratégies de gestion de ressources capables d'atteindre ces 2 objectifs presque conflictuels.

Dans le cadre de cette thèse, nous nous intéressons à la gestion du CPU en environnement virtualisé, en prenant soin de garantir l'accord de niveau de service

des clients et de permettre une utilisation optimale des ressources en terme d'énergie consommée. En effet, avec l'adoption du principe de variation de vitesse d'exécution des périphériques lors de la mise en œuvre des politiques de gestion d'énergie, les machines virtuelles qui en ont besoin, reçoivent moins que la capacité souscrite. Afin d'y faire face, l'une des méthodes consiste à augmenter (respectivement diminuer) la capacité à allouer aux machines virtuelles lors des baisses (respectivement hausses) de vitesse d'exécution.

Dans cette thèse, nous définissons les principes de conception d'un gestionnaire de ressources favorisant la réduction de l'énergie consommée par le périphérique tout en respectant l'accord de niveau de service des machines virtuelles qui l'utilise. Nous proposons un prototype de gestion de CPU (ordonnanceur de machines virtuelles), nommé *pas-sla-scheduler*, qui se sert du principe de périphérique à vitesse variable afin de réduire la consommation énergétique du CPU. De plus, afin de garantir l'accord de niveau de service des machines virtuelles, notre ordonnanceur *pas-sla-scheduler* augmente (respectivement diminue) dynamiquement la capacité CPU allouée aux machines virtuelles si la vitesse d'exécution de la ressource est diminuée (respectivement augmentée).

Ce document est organisé de la façon suivante :

– **Première partie : contexte d'étude**

  Cette partie présente le contexte général de nos travaux de recherche. Elle est composée des chapitres 2 et 3. Dans le chapitre 2, nous présentons le contexte scientifique de cette thèse qui est le cloud computing, et les raisons ayant favorisées son expansion. Au terme de ce chapitre, nous identifions la gestion d'énergie comme un défi majeur pour un fournisseur de services. Dans le chapitre 3, nous définissons un modèle de gestion d'énergie basé sur la variation de la vitesse d'exécution des périphériques et, effectuons un état de l'art de quelques solutions de gestion d'énergie suivant ce modèle.

– **Deuxième partie : Problématique et état de l'art**

  Cette partie est destinée à définir la problématique que nous abordons durant nos travaux. Elle présente également quelques travaux de recherche (les plus représentatifs) liés à cette problématique. Cette partie est composée des chapitres 4 et 5. Le chapitre 4 est subdivisé en 2 grandes sections. La première section rappelle les exigences d'un gestionnaire de ressources dans un IaaS, à savoir le respect de l'accord de niveau de service et l'utilisation optimale des ressources. Dans cette même section, nous montrons les limites des approches actuelles de gestion de ressource au regard de ces exigences. Dans la seconde section, nous considérons le cas particulier du CPU, et relevons les faiblesses des ordonnanceurs de machines virtuelles actuels. Dans le chapitre 5, nous définissons des critères d'évaluation de solutions et, effectuons un état de l'art de quelques travaux de recherches, liés à cette problématique, sur la base de ces

critères.

– **Troisième partie : Contributions**

Cette partie est destinée à la présentation de notre approche, son implémentation et son évaluation. Elle est divisée en 3 chapitres 6, 7 et 8. Le chapitre 6 présente l'orientation générale et la conception adoptées pour la mise en place d'un ordonnanceur de machines virtuelles respectant les 2 objectifs sus-cités. Dans le chapitre 7, nous décrivons plus en détail l'implémentation de notre ordonnanceur *pas-sla-scheduler*. Le chapitre 8 présente les résultats des évaluations que nous avons effectuées. Nous présentons trois groupes d'évaluation : (1) des mesures permettant de vérifier l'ensemble des hypothèses de proportionnalité que nous avons défini lors de l'implémentation de notre ordonnanceur ; (2) une évaluation qui certifie les limites des solutions actuelles d'ordonnancement de machines virtuelles ; (3) une évaluation permettant de vérifier que notre approche est conforme à ces 2 critères.

Nous concluons notre document au chapitre 9 et présentons quelques perspectives liées à la mise en œuvre de notre approche.

Première partie

Contexte d'étude

Chapitre 2

Contexte général

Contents

Ces dernières années ont été marquées par une hausse du coût de fonctionnement (mise en place et maintenance) des infrastructures informatiques quelle que soient leur taille (PME[1], centre d'hébergement). Dès lors, de multiples méthodes ont été proposées afin d'y faire face. L'une des plus récentes est le *cloud computing*.

Le *cloud computing* constitue le contexte général de nos travaux de recherche. De ce fait, dans ce chapitre subdivisé en 4 sections, nous présentons : (1) le coût

1. Petite et Moyenne Entreprise

élevé du fonctionnement des infrastructures informatiques comme l'une des raisons ayant motivée l'adoption du cloud, (2) les caractéristiques du cloud et les atouts qu'il apporte pour la réduction de ces coûts, (3) les différents modèles de service qu'offre le cloud ainsi que les technologies utilisées pour leur mise en œuvre et enfin (4) les défis que soulève la mise en place du cloud.

2.1 Coût d'une infrastructure informatique

La mise en place et la gestion locale d'un service informatique dans la plupart des petites/moyennes/grandes entreprises s'avèrent extrêmement coûteuses. Ces coûts sont généralement de divers ordres : matériel, logiciel, humain et énergétique.

2.1.1 Coût matériel

Le coût matériel d'une infrastructure informatique dépend du nombre d'équipement qu'elle contient. Dans un service informatique, l'acquisition des serveurs se fait de façon progressive. Dans le souci de garantir une haute disponibilité des services et de mettre en place un système de reprise après panne, la majorité de ces serveurs est déployée avec redondance. De plus, dans le but de pouvoir répondre aux fortes montées de charges et de garantir un temps de réponse optimum, l'allocation et le déploiement des ressources ont tendance à être sur-dimensionné. Dans ce contexte, les serveurs sont en général chargés en moyenne à 20% [134] au sein des entreprises. Ce qui entraine en conséquence un gaspillage de ressources et une augmentation inutile du coût matériel des infrastructures informatiques.

2.1.2 Coût logiciel

Le coût logiciel d'une infrastructure est généralement proportionnelle à la taille de l'infrastructure matérielle. En effet, ce coût logiciel englobe le coût d'achat des licences, souvent facturé au nombre d'équipement sur lequel il est installé et/ou au nombre de CPU de l'équipement, le coût d'achat éventuel des mises à jour ou des fonctionnalités optionnelles et le coût du support. De ce fait, plus l'infrastructure matérielle est grande plus le coût logiciel ira croissant.

2.1.3 Coût humain

Le coût humain d'une infrastructure informatique renvoie au personnel en charge de son administration (et sa maintenance) et est fonction de l'architecture [2] de l'infrastructure. L'infrastructure informatique d'une entreprise constitue en général un environnement hétérogène et complexe. Hétérogène et complexe du fait de l'achat progressif des équipements, du déploiement et de l'utilisation de différents systèmes d'exploitation (OS [3]), d'applications et d'outils d'administration [14]. Cette hétérogénéité a pour conséquence (1) de nécessiter différentes formations en vue d'un personnel qualifié et (2) une faible productivité du personnel informatique. En effet, elle nécessite parfois qu'une grande majorité (au minimum 70%) des ressources humaines soit dédiée à l'administration et à la maintenance des systèmes existants plutôt qu'au développement de nouvelles compétences informatiques [119]. Ainsi, plus l'infrastructure sera grandissante, plus l'entreprise aura besoin de personnel et sera sujette à effectuer de nouvelles dépenses liées aux formations.

2.1.4 Coût énergétique

Aux différents coûts sus-cités, s'ajoute le coût énergétique qu'engendre cette infrastructure. Les rapports scientifiques réalisés au cours de ces 20 dernières années montrent que les infrastructures informatiques des entreprises sont de plus en plus consommatrices en énergie [66]. Concrètement, l'énergie consommée se compose de l'énergie utile au fonctionnement de l'infrastructure matérielle (serveurs, équipements de stockage, équipements réseaux et de télécommunication) et de l'énergie consommée par l'infrastructure dédiée au refroidissement, à la ventilation, à l'éclairage, aux onduleurs, pour ne citer que ceux-là. A cause de la redondance des services, l'infrastructure informatique représente à elle seule plus de 50% de l'énergie consommée au sein de certaines catégories d'entreprises [67].

En somme, le coût de gestion locale d'une infrastructure informatique est enclin à de fortes hausses. Fort de ce constat, plusieurs entreprises ont entrepris de l'externaliser.

2. Dans notre contexte, l'architecture d'une infrastructure fait référence à *l'architecture logicielle* (une vue tournée sur l'organisation interne et le découpage en couches et modules du ou des logiciels du système informatique) et *l'architecture technique* (une vue tournée vers les différents éléments matériels et l'infrastructure dans laquelle le système informatique s'inscrit, les liaisons physiques et logiques entre ces éléments et les informations qui y circulent)

3. Operating System

2.2 Externalisation et accord avec le fournisseur

2.2.1 Choix d'externaliser

Au regard du gaspillage des ressources dû au sur-dimensionnement, à la sous utilisation des ressources [10] et aux différents coûts sus-cités, de nombreuses entreprises ont recours à l'externalisation de leur infrastructure informatique. Cette dernière consiste à déléguer certaines des tâches de l'entreprise à un prestataire externe et à focaliser l'ensemble des ressources sur des tâches ayant une forte valeur ajoutée pour l'entreprise. Dans ce but, l'entreprise a besoin d'un personnel minimal et qualifié chargé de déterminer le nombre et la capacité des serveurs capables de traiter de manière satisfaisante leurs opérations [126]. Après avoir identifié ses besoins, l'entreprise souscrit pour un service auprès d'un *fournisseur de services* (que nous nommerons *fournisseur*).

2.2.2 Nature du fournisseur

Traditionnellement, le fournisseur est représenté par un centre d'hébergement (*Hosting Center* en anglais). Il se caractérise par un lieu où se trouvent différents équipements électroniques, des ordinateurs inter-connectés (cluster), des systèmes de stockage et des équipements de réseaux et télécommunication. Les serveurs sont accessibles à distance et la mise à la disposition des clients est effectuée suivant 2 modèles d'allocation : *statique* et *dynamique* :

1. ***Modèle d'allocation statique***. Dans ce modèle d'allocation, la ressource allouée reste inchangée durant toute la période d'utilisation. Ce modèle d'allocation comporte 2 politiques d'allocation de ressources : *allocation dédiée* et *allocation d'une portion.*
   - ***Allocation dédiée*** : en fonction des caractéristiques matérielles souhaitées par le client, le fournisseur lui alloue de façon exclusive un ensemble de serveurs satisfaisant sa demande. Ce choix d'allocation a l'avantage de permettre la réduction des coûts énergétiques (ventilation, onduleur, etc) et de maintenance des entreprises.
   - ***Allocation d'une portion*** : le pourcentage d'utilisation des ressources d'un serveur étant généralement faible [4], la pratique d'allocation dédiée se traduit par un gaspillage de ressources au niveau du fournisseur et/ou un sur-coût pour le client. Afin de réduire ce gaspillage et accroître l'utilisation de ses ressources, le fournisseur implante une politique d'allocation qui consiste à allouer des portions de serveurs (*unité d'allocation plus fine*), au lieu d'allouer un ensemble de serveurs, à ses clients.

4. Par exemple, le pourcentage d'utilisation du processeur d'un serveur n'excède presque pas (pour la plupart) 20%

2. ***Modèle d'allocation dynamique***. La mise en place de l'allocation statique (présenté plus haut) nécessite de connaître à l'avance les besoins des clients, afin d'éviter les situations de gaspillage de ressources dû au sur-dimensionnement ou de perte de performance à cause du sous-dimensionnement. Le présent modèle d'allocation permet ainsi d'allouer et de libérer dynamiquement les ressources en fonction de la charge.

2.2.3 Caractéristiques du Cloud

Au milieu de la pléthore de définitions du cloud, nous adoptons celle de l'organisme de normalisation *National Institute of Standards and Technology (NIST)* que nous traduisons comme suit : le *cloud* est un modèle de service permettant un accès partagé, à la demande, via un réseau de télécommunications, à des ressources informatiques configurables (réseaux, serveurs, stockage, applications et services) [90]. Ces ressources peuvent être rapidement approvisionnées et libérées avec un effort de gestion minimal ou l'intervention du fournisseur de service.
Cette définition met en évidence une des principales caractéristiques du cloud : **l'allocation à la demande** (communément appelé **allocation dynamique** dans la littérature). L'allocation à la demande des ressources repose sur une distribution dynamique de ressources entre utilisateurs. En effet, le fournisseur dispose d'une ressource limitée qu'il alloue parallèlement et/ou successivement entre ses clients en réponse aux différentes demandes reçues. Le client souscrit pour une ressource maximale qui ne lui est allouée qu'en cas de nécessité. Cette ressource est automatiquement adaptée en cours d'exécution en fonction des besoins et sur demande du client.
Après utilisation, la ressource est libérée et peut servir à répondre à la demande d'un autre utilisateur. Du fait que les ressources puissent ainsi être libérées et allouées à d'autres clients après leur utilisation, le cloud donne l'illusion d'une ressource illimitée. L'allocation à la demande permet donc d'absorber instantanément les pics de charges et de libérer les ressources pour répondre aux besoins d'autres clients.

L'allocation à la demande donne lieu à une autre caractéristique du cloud relative à son principe de facturation. Le service de facturation applicable dans un cloud lui est spécifique et est lié à son principe de fonctionnement. Il est connu sous le nom de *pay-as-you-go* [86, 52]. Théoriquement, le client ne paye que pour les services (matériel et logiciel) réellement utilisés sans se préoccuper des détails de coût de fonctionnement et des opérations d'administration (mises à jour, installation des patchs) [4, 90].

2.2.4 Atout du Cloud pour les entreprises

Les caractéristiques associées au cloud ci-dessus présentées permettent aux entreprises de réduire le coût total de possession des systèmes informatiques. En

effet, la mise en œuvre de l'allocation à la demande résout le problème de surdimensionnement et de gaspillage de ressources. Elle permet aussi de pallier aux éventuelles saturations dues au sous-approvisionnement parce qu'elle offre la possibilité d'augmenter ou de diminuer dynamiquement et rapidement les ressources. La facturation basée sur l'utilisation effective des ressources permet aux entreprises de ne payer que pour ce qu'elles ont véritablement besoin, ce qui contribue également à la baisse du coût général du système informatique.

2.2.5 Accord entre client et fournisseur (SLA)

Le contrat qui lie le client et le fournisseur se définit sous forme de qualité de service (QoS pour *Quality of Service*) à garantir. Les conditions de satisfaction et les pénalités applicables au fournisseur en cas de non respect de cette QoS sont consignées dans un document sous le nom d'*accord de niveau de service* (Servive Level Agreement - SLA- en anglais) [68].
La QoS désigne la capacité d'un service à répondre par ses caractéristiques aux différentes exigences de ses utilisateurs. Dans le cloud, la QoS dépend du modèle de service (que nous détaillons en section 2.3) souscrit par le client. Si le client souscrit pour l'utilisation :

- d'un logiciel : la QoS peut par exemple s'exprimer en terme de disponibilité (Ex : taux de rejet), fiabilité (Ex : temps moyen entre deux pannes), performance (Ex : temps de réponse) ou volume de transactions effectuées par minute, etc [19] ,
- d'une plateforme de déploiement : elle peut s'exprimer sous forme de version de logiciel, d'environnement de développement ou de compatibilité entre les technologies,
- d'une infrastructure : le client spécifie les caractéristiques matérielles souhaitées. Elles sont exprimées sous forme de capacité d'unité de calcul (CPU [5]), de quantité de mémoire (RAM [6]), d'espace de stockage des données et de bande passante du réseau [8, 153].

Le fournisseur est chargé de veiller au respect de cette QoS. Il est de ce fait responsable de définir une architecture lui permettant de satisfaire aux exigences des clients.

5. Central Processor Unit
6. Random Access memory

2.3 Architecture d'une infrastructure de cloud

2.3.1 Modèle de service du Cloud

Le cloud désigne à la fois les logiciels livrés en tant que service via internet, les plateformes de déploiement d'applications et l'infrastructure qui fournit ces services [3]. Son architecture est fonction du public cible et du type de service offert. Ainsi, sur la base du public concerné, NIST définit 3 familles de cloud. Un cloud destiné à l'usage restreint d'une entreprise porte le nom de *cloud privé* [7]. Dans le cas contraire, c'est-à-dire ouvert au grand public, il porte le nom de *cloud public* [8]. Il existe des situations pour lesquelles les utilisateurs de cloud privé nécessitent l'usage d'un cloud public pour des besoins précis. Cette configuration décrit une nouvelle forme de cloud connue sous le nom de *cloud hybride* [9] [90, 73].

De même, en fonction du service offert et du niveau d'abstraction de celui-ci, NIST définit 3 modèles de service que peuvent offrir le cloud [55, 90, 143] :

- *Software-as-a-Service (SaaS)* : modèle dans lequel les services proposés sont des logiciels mis à la disposition des clients et accessibles via Internet. Le client n'a aucune responsabilité au sujet de l'infrastructure sous-jacente du cloud à l'exception des paramètres de configuration spécifiques à l'utilisateur ;
- *Plateform-as-a-Service (PaaS)* : modèle dans lequel un environnement de travail est fourni au client. Ce dernier a la capacité de déployer des applications, créées ou acquises, en utilisant des langages de programmation, des bibliothèques, des services et des outils supportés par le fournisseur. Le client n'a aucune responsabilité sur l'infrastructure de cloud sous-jacente. Cependant, il a le contrôle sur les applications déployées et les paramètres de configuration possible pour l'environnement d'hébergement des applications ;
- *Infrastructure-as-a-Service (IaaS)* : modèle qui consiste à offrir à un client des machines virtuelles (VM [10]), caractérisées par une puissance de calcul, un espace de stockage, des accès réseaux, et d'autres ressources informatiques fondamentales afin que l'utilisateur soit en mesure de déployer et d'exécuter tout logiciel (OS et applications). Le client ne s'occupe pas de la gestion de l'infrastructure cloud sous-jacente, mais exerce un contrôle sur l'OS, le stockage, les applications déployées, et éventuellement un contrôle limité des composants réseaux utilisés (par exemple un pare-feu).

D'autres types de service (Business Process, Network, Database, etc) peuvent également être offerts via un cloud. Au regard de la diversité de ceux-ci, les modèles de service de cloud sont généralement regroupés sous le nom de *Anything-as-a-Service ou X-as-a-Service (XaaS)* [68, 115, 76].
Salesforce.com [Salesforce], Google App Engine [Google] et Amazon Elastic Compute

7. *Private cloud*
8. *Public cloud*
9. *Hybrid cloud*
10. Virtual Machine

Cloud (EC2) [Amazon] constituent 3 exemples de fournisseurs de SaaS, Paas et IaaS respectivement.

Suivant la hiérarchie *bottom-up* [11], le IaaS constitue la couche la plus basse. Elle permet l'implémentation des services des couches supérieures. Fort de cet atout, dans la suite de ce document (également dans le cadre de nos travaux de recherche), nous nous intéressons uniquement au IaaS [12] ; plus particulièrement aux technologies de virtualisation (section suivante) qui servent à son implémentation et aux défis que soulèvent sa gestion.

2.3.2 Technologie de virtualisation

Définition

La *virtualisation* se définit comme l'ensemble des techniques matérielles et/ou logicielles qui permettent de faire fonctionner sur une seule machine physique différents OS [39]. Séparés les uns des autres (Figure 2.1), l'utilisation de ces différents OS donne à l'utilisateur l'illusion d'être en présence de plusieurs machines distinctes. Concrètement, une couche d'abstraction logicielle communément appelée *logiciel de virtualisation* (*VMM* [13]) ou *hyperviseur* est installée dans le *système hôte (host)* (OS installé sur la machine physique) et est chargée de créer des *machines virtuelles (VM* [14]*)* sur lesquelles sont installées les *OS invités* (*guest*).

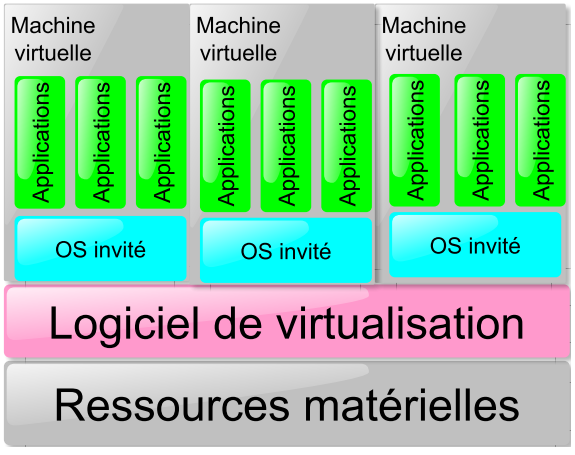

Fig 2.1. Architecture virtualisée

11. De bas en haut
12. Nous utilisons indifféremment le mot cloud ou IaaS dans la suite du document
13. Virtual Machine Monitor
14. Virtual Machine

Principe de fonctionnement

En plus d'être chargée de la création des VMs, la VMM a pour rôle de partager et de coordonner les accès aux ressources de l'hôte physique sous-jacent entre les OS invités. Bien qu'ils soient virtuels, une VM dispose d'un ensemble de périphériques. Une VM possède ainsi une puissance de calcul (nombre et capacité du CPU), un espace de stockage (en général représenté par des fichiers sur les systèmes de stockage), une quantité de mémoire et une ou plusieurs cartes réseaux. Au cours de son fonctionnement, tout accès de l'OS invité à un périphérique virtuel est géré par la VMM qui se charge d'interpréter et d'exécuter la demande sur le périphérique réel sollicité.

Cette architecture et ce mécanisme de translation sont entièrement transparents à l'utilisateur final qui s'imagine disposer d'une machine physique réelle. Ceci est rendu possible grâce au respect des critères suivants établis par Popek et Goldberg [112] pour les VMM :

- *équivalence* : toute exécution d'application dans un système virtualisé doit être identique à une exécution sur une machine physique ; exception faite du temps d'exécution lié à la disponibilité des ressources,
- *efficacité* : la majorité des instructions du système invité doit directement être exécutée par le processeur sans intervention du logiciel de virtualisation,
- *contrôle de ressources* : l'ensemble des ressources est géré de façon exclusive par le logiciel de virtualisation ;

Ainsi, les applications déployées dans les VMs sont exécutées de façon identique à une exécution sur un support physique et ne nécessitent aucune modification.

Atouts de la virtualisation

Les VMs s'exécutant sur le même support matériel, la virtualisation accorde un grand intérêt à l'*isolation* ; aussi bien à l'isolation des pannes qu'à l'isolation de performance ; et à la *sécurité* des données. L'isolation des pannes permet d'éviter que la compromission d'une VM ne se propage. Pour cette raison, la virtualisation intègre des techniques d'encapsulation. Ainsi, les OS invités (ainsi que les applications qui y sont installées) sont encapsulés et cloisonnés au sein de VMs de telle sorte que la corruption d'une application (dans une VM) n'influence aucunement le bon fonctionnement d'autres logiciels présents sur le système hôte. L'isolation de performance est également assurée par l'encapsulation. Cette dernière garantit qu'aucune sur-exploitation de ressources par une VM ne prive d'autres de ce qui leur est réservé.

La virtualisation offre également la possibilité de réduire les coûts énergétiques. Grâce à la fonctionnalité de migration de VMs [22] qu'intègre la virtualisation, il est possible de tasser dynamiquement (consolider) de nombreuses VMs au sein d'un même ordinateur physique. La mise en place de cette opération de consolidation veille au préalable à ce que cet ordinateur soit capable de supporter la charge générée

par les VMs. Cette possibilité de migration et de consolidation permet de réduire le nombre de machines physiques en fonctionnement. La réduction du nombre de machines physiques conduit à réduire les coûts énergétiques liés à leur exploitation et refroidissement.

Types de technologie de virtualisation

Il existe plusieurs familles de solutions de virtualisation : *virtualisation totale*, la *paravirtualisation* et la *virtualisation assistée par le matériel*[15] *(HVM*[16]*)*.

La virtualisation totale consiste à émuler entièrement le matériel du système hôte pour le rendre accessible à une VM. Les accès aux périphériques s'effectuent comme si la VM était en présence d'un matériel réel. La VMM se charge d'effectuer les opérations nécessaires pour translater (convertir) les instructions privilégiées de l'OS de la VM en opérations compréhensibles et réalisables sur le support physique de l'OS hôte. Dans cette forme de virtualisation, l'OS invité et les applications qui y sont installées ne requièrent aucune modification ou adaptation. Toutefois, cette solution de virtualisation est sujette à une perte de performance importante due au nombre de couches logicielles "à traverser" lors du traitement de toute requête. Cette forme de virtualisation est implémentée entre autres par VMware Workstation [146, 129], Qemu [7, 6].

La paravirtualisation a été proposée afin de résoudre les problèmes de perte de performance observés dans l'utilisation de la virtualisation totale. La paravirtualisation consiste à modifier l'OS invité de façon à lui inclure des instructions spéciales nommées *hypercalls*. Ces dernières permettent des accès aux périphériques sans avoir recours aux translations. Cette forme de virtualisation a l'avantage d'offrir des performances d'applications exécutées dans les VMs semblables aux performances obtenues lors d'une exécution sur un système non virtualisé (aussi appelé *système natif*). Cette technologie a été principalement implémentée par Xen [12, 21] et VMware ESXi [141, 91, 85]. Toutefois, cette technologie de virtualisation exige de disposer du code source de l'OS invité. Cette contrainte limite l'exploitation de la paravirtualisation uniquement aux OS non propriétaires.

L'avènement de processeurs intégrant des instructions de virtualisation a fait naître une nouvelle famille de technologies de virtualisation : *la virtualisation assistée par le matériel (HVM)*. Elle consiste à gérer l'ensemble des opérations de virtualisation de façon matérielle et non plus logicielle comme dans les solutions précédentes. Cette forme de virtualisation offre des performances de VM identiques aux OS natifs mais a pour contrainte de disposer d'un matériel intégrant les instructions de virtualisation. Intel et AMD grâce à leurs technologies VT-x (Virtual Technology) [131, 135, 48] et SVM(Secure Virtual Machine)/Pacifica [140, 116] res-

15. Hardware-Assisted Virtualization
16. Hardware Virtual Machine : terminologie de Xen

pectivement, offrent ces instructions et sont exploitables par VMware ESXi [141, 91], Xen [12, 152], KVM [63, 46, 120], VirtualBox [139, 147, 102, 103] ou Hyper-V [137, 75, 114].

2.3.3 Infrastructure-as-a-Service

Le modèle de service IaaS consiste à mettre à la disposition des utilisateurs un parc informatique virtualisé. Comme toute infrastructure, son fonctionnement efficace est conditionné par l'ensemble des opérations d'administration effectuées. Pour ce faire, le IaaS intègre en plus des serveurs de stockage des images de VM, des services de réservation de ressources, de monitoring et d'équilibrage de charge entre les serveurs.

Réservation de ressources

. Elle consiste à allouer au client la ressource (CPU, RAM, espace disque, bande passante,etc) demandée. La demande est généralement effectuée via une réservation. Le fournisseur, en fonction des stratégies de gestion de son infrastructure, propose plusieurs formes de réservation : *réservation immédiate* (le besoin en ressource doit immédiatement être satisfait), *réservation programmée* (le client indique la quantité de ressource souhaitée, la date de disponibilité de la ressource et la durée d'utilisation). Ainsi au déclenchement de la réservation, les VMs sont démarrées et toutes les opérations de configuration permettant leur accès et leur exploitation à distance par le client sont effectuées. Les ressources sont libérées au terme de la réservation. Toutefois, le client peut établir un contrat à durée indéterminée avec le fournisseur et avoir ainsi immédiatement accès aux ressources en cas de besoin.

Monitoring et équilibrage de charge [17]

. Le gestionnaire de ressources dans un IaaS se sert d'un système de monitoring pour prendre des décisions avisées lors du déploiement d'une nouvelle VM ; en cas de détection de pannes ou pour pallier à la perte de performances des VMs.
Par défaut, le monitoring donne une vue globale du système et permet d'identifier la machine physique sur laquelle la VM est initialement exécutée. Ce choix est en général fonction des critères de gestion du fournisseur [92]. Le monitoring permet aussi de déceler un dysfonctionnement dans l'infrastructure et d'agir en conséquence. Notamment, la détection d'une panne peut conduire au redémarrage d'une VM ou à la sauvegarde de son état courant grâce à la fonctionnalité de sauvegarde (checkpointing) [24] qu'apporte la virtualisation.

17. Load balancing

En fonction de l'état d'occupation des machines de l'infrastructure du IaaS, le service d'équilibrage de charge peut effectuer des migrations de VMs (atout qu'apporte la virtualisation). Cette opération permet de libérer des machines sous chargées et les éteindre éventuellement en vue de réduire la consommation énergétique de l'infrastructure. Cette action permet aussi d'alléger certaines machines afin d'éviter les pertes de performance des VMs en cas de saturation/surcharge.

Le fournisseur exploite de cette manière les fonctionnalités d'encapsulation et de cloisonnement pour assurer l'isolation des ressources entre les clients et garantir la sécurité des données des différentes applications co-hébergées. Grâce au checkpointing et à la migration des VMs, le fournisseur essaie de préserver le niveau de performance souscrit par le client et de réduire l'énergie que consomme son infrastructure.
Pour les entreprises, l'utilisation du cloud et son principe de facturation à l'usage favorisent la réduction des coûts de gestion locale des systèmes informatiques. Toutefois, la mise en place d'un cloud présente de nombreuses contraintes et son fonctionnement efficace engendre de nombreux défis côté fournisseur.

2.4 Défis liés à l'utilisation d'un IaaS

De nos jours, le cloud peut être exploité dans de nombreux domaines tels que la finance, le commerce électronique (*e-commerce*), l'industrie ou la recherche scientifique [109, 15]. Ceci se justifie par le fait que les utilisateurs ont à leur disposition une infrastructure de pointe et performante sans avoir à faire de lourds investissements et à prendre en compte l'administration et la maintenance de celle-ci. Malgré cela, l'adoption du cloud suscite des craintes liées à la sécurité, à la confidentialité des données, au respect du SLA et à l'énergie qui illustrent les principaux défis que rencontrent les fournisseurs.

2.4.1 Sécurité et confidentialité

Les données des clients d'un IaaS sont stockées de façon distribuée, entre les serveurs d'un même site ou entre les serveurs de différents sites, et accessibles via Internet. Le fournisseur doit assurer la sécurité, la confidentialité et l'intégrité des données ainsi gérées. Pour ce faire, il utilise des techniques d'authentification et de cryptographie pour contrôler et filtrer l'accès et l'utilisation des ressources [11, 5, 108, 151, 50, 113]. D'autres mécanismes de sécurité employés par les fournisseurs intègrent l'utilisation des VLAN regroupant les VMs d'un même client.

2.4.2 Respect du SLA

Lors de la souscription d'un service, le client spécifie les caractéristiques matérielles des VMs souhaitées et leur respect constitue l'un des défis présents dans le IaaS.
Afin de garantir ce SLA, le gestionnaire de ressources dans un IaaS alloue dynamiquement les ressources aux clients et effectue au besoin des opérations d'équilibrage de charge afin d'éviter tout abus des clients. L'équilibrage de charge se doit d'éviter toute situation où certains serveurs sont surchargés tandis que d'autres sont en veille ou sous-chargés [51].

2.4.3 Energie

Dans un IaaS, l'ensemble des ressources matérielles ne sont pas toujours utilisées au maximum de leur performance. En cas de forte montée de charge, un grand nombre de machines sont sollicitées et utilisées intensément. Au terme de ces piques de charge, les machines sont en grande majorité sous utilisées. Étant donné que le fournisseur ne peut prévoir en temps réel [18] l'instant et le débit d'arrivée des accès aux VMs qu'il héberge, celui-ci est tenu de conserver son infrastructure en continuel état de marche. Ceci lui permet de satisfaire rapidement d'éventuelles et subites montées de charge.
Cependant, le fait de laisser des machines fonctionner au meilleur de leur performance en situation de faible charge conduit à un gaspillage énergétique préjudiciable pour le fournisseur. De ce fait, en plus des défis liés à la sécurité et au respect du SLA (ci dessus présentés), le fournisseur doit faire face à une gestion efficiente du coût énergétique qu'engendre son infrastructure notamment en réduisant le nombre de serveurs en état de marche(que nous détaillons au chapitre 3).

2.5 Synthèse

Dans cette section, il était question du contexte général encadrant nos travaux. Nous avons étudié les raisons qui justifient le coût financier élevé qu'occasionne la gestion locale de l'infrastructure informatique de certaines entreprises. Ces raisons s'articulent autour des coûts du matériel, des logiciels, du personnel et de la consommation énergétique de l'infrastructure. Ces éléments constituent une explication à la forte externalisation des services informatiques auprès des fournisseurs. Nous avons également fait état des caractéristiques fonctionnelles de ces fournisseurs de service tout en montrant en quoi leur adoption favorise la réduction des coûts des systèmes informatiques des entreprises.

18. Une prévision est possible en postériori en se basant sur une analyse "offline" des données

Bien que l'externalisation des services soit avantageuse pour certaines entreprises, elle soulève de nouveaux défis côté fournisseur. Le premier défi porte sur les données et est en lien avec leur sécurité, leur confidentialité et leur intégrité. Ce défi est relevé grâce à la mise en place des mécanismes d'authentification, de filtre et de cryptage de données. Le second challenge concerne le SLA. Ce dernier est intimement lié à la QoS dans un IaaS. Par conséquent, les opérations effectuées pour un équilibrage de charge doivent à terme fournir cette même QoS. Un troisième défi, celui relatif à la consommation énergétique demeure présent pour le fournisseur.

Dans le chapitre suivant, nous examinons les politiques existantes susceptibles de permettre au fournisseur de réduire au mieux le coût énergétique de son infrastructure. Dans le cadre de cette thèse, nous nous focalisons sur ce défi des fournisseurs face à la consommation énergétique de leur infrastructure.

Chapitre 3

Gestion d'énergie dans le Cloud

Contents

Dans le chapitre précédent, nous avons présenté les facteurs ayant contribué à l'externalisation de l'infrastructure informatique de certaines entreprises. Nous avons mis en évidence les 3 principaux défis actuels que rencontrent les fournisseurs de cloud. Les 2 premiers (sécurité des données et SLA) sont en général satisfaits grâce aux mécanismes d'authentification, de filtrage, de cryptage et d'équilibrage de charge appliqués dans le IaaS. Le troisième défi, portant sur la réduction du coût de la consommation énergétique fait l'objet de ce chapitre.

Ce chapitre est subdivisé en 3 grandes sections. La première section définit et présente un modèle de gestion d'énergie applicable sur les principaux périphériques d'un ordinateur. Sur la base de ce modèle, nous étudions dans la deuxième section quelques politiques de gestion d'énergie basées sur ce modèle. Dans la dernière section de ce chapitre, nous examinons de façon critique l'impact positif et/ou négatif de ces politiques de gestion d'énergie appliquées dans un contexte de systèmes virtualisés.

3.1 Modèle de gestion d'énergie

Pour espérer une diminution de la consommation énergétique de son infrastructure, le fournisseur est tenu de définir et de mettre en place des politiques de gestion d'énergie efficaces.
Dans cette partie, nous définissons un ***modèle de gestion d'énergie***. Il s'agit d'un découpage "sémantique" des politiques de gestion dynamique d'énergie les plus souvent utilisées. Dans ce modèle, nous nous basons sur un couple formé d'une ressource et des opérations applicables sur celle-ci. La ressource dans notre contexte représente tout équipement électronique en production au sein d'un IaaS quel que soit son état (présenté en section 3.1.1). Les opérations quant à elles représentent toute action permettant de faire varier la vitesse d'exécution de cette ressource (présenté en section 3.1.2).

3.1.1 Etat d'une ressource

Afin de mieux comprendre le principe de variation de vitesse d'exécution d'une ressource, nous définissons quelques concepts clés, relatifs à l'état d'une ressource, que nous utilisons par la suite. A l'image du fonctionnement standard d'un processus, la majorité des ressources informatiques subissent, au cours de leur exécution, des changements d'état que nous illustrons sous forme d'un diagramme d'état-transition (Figure 3.1).

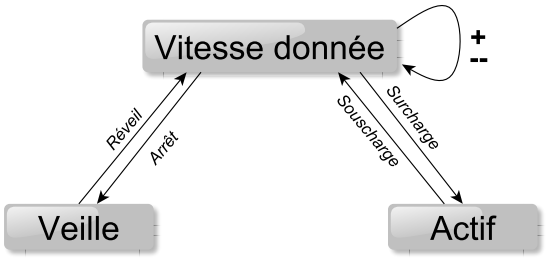

Fig 3.1. Diagramme de changement d'état d'une ressource

Une ressource peut ainsi être dans un état : *actif, veille ou de vitesse donnée* :
- ***État actif*** : la ressource est en cours d'exécution. Par défaut, elle fonctionne au meilleur de ses performances et consomme par conséquent l'énergie qui lui est nécessaire (d'après les normes du constructeur),
- ***État de veille*** : la ressource est "endormie". Elle nécessite un temps d'activation relativement long, dû au "réveil" de la ressource, en cas de nouvelle requête. La ressource n'est ni en fonctionnement et ni en attente active d'une requête. Cet état représente l'état le plus économique en énergie,
- ***État de vitesse donnée*** : la ressource est en cours d'exécution et sa vitesse est susceptible de varier (situation semblable à un ralentissement ou à une

accélération). Elle peut ainsi fonctionner à différentes vitesses d'exécution et possède de ce fait un niveau de performance variable (fonction de sa vitesse courante). Ces changements sont soit le résultat d'une opération de configuration déclenchée par un acteur externe soit l'objet d'une politique implantée dans le périphérique. Peu importe l'origine de cet état, la ressource consomme une quantité d'énergie en général proportionnelle [1] au niveau de performance d'exécution.

Comme exemple de politique, nous observons qu'au terme de l'exécution d'une opération et passé un délai d'attente initialement prédéfini [2], la ressource peut passer de l'état actif à un état de vitesse inférieure puis de veille successivement. Elle ne revient à l'état actif qu'à l'arrivée de nouvelles opérations à exécuter.

Les opérations de variation de vitesse que nous définissons dans notre modèle sont essentiellement construites autour de ce diagramme d'état.

3.1.2 Opérations de variation de vitesse d'exécution

Dans le modèle que nous définissons, les opérations de variation de vitesse d'une ressource sont possibles soit (1) nativement de par la conception de la ressource [3], soit (2) sont simulées à travers le regroupement des traitements qu'effectue la ressource.

3.1.2.1 Variation de vitesse de façon native

Cette famille d'opérations tire parti du fait que le périphérique/composant, de par sa conception, est capable d'être dans l'un des états actif, veille ou vitesse donnée. Ainsi, en se basant sur le taux d'utilisation du périphérique/composant, les changements d'états (nativement intégrés dans le périphérique/composant) permettent de réduire la consommation énergétique de la ressource en l'arrêtant (*état de veille*) ou en réduisant sa vitesse d'exécution (*état de vitesse donnée*) lorsqu'elle est inactive ou sous utilisée respectivement [9]. L'*état de vitesse actif* étant utilisé quand le périphérique/composant est surchargé.

3.1.2.2 Regroupement des traitements

Toutefois, le changement de vitesse d'exécution de certains périphériques/composants en fonction de leur charge n'est toujours pas existant (de façon native).

1. Proportionnalité pas forcement linéaire
2. D'autres politiques de gestion du délai peuvent également être mise en œuvre.
3. By design

De ce fait, plusieurs approches ont entrepris de l'émuler de façon logicielle grâce au *regroupement (temporel ou spatial) des traitements* :

- Lors d'un ***regroupement temporel*** des traitements, le périphérique/composant est mis en état de veille pendant toute une période de temps dédiée à l'accumulation (*au buffering*) des opérations à traiter. A l'échéance d'un délai (fixe ou variable), le périphérique/composant passe à l'état actif et traite l'ensemble des requêtes en attente.
- Le ***regroupement spatial*** s'applique sur un ensemble de périphériques sur lesquels la charge est repartie.

Ces mécanismes permettent de simuler l'exécution du périphérique/composant à une vitesse donnée. Cette vitesse représente, en théorie, la vitesse moyenne d'exécution du périphérique/composant pendant toute la période de traitement. Ces simulations permettent ainsi de maximiser le rendement des périphériques/composants en terme de consommation énergétique.

Somme toute, il serait logique d'affirmer que les solutions de gestion d'énergie consistent toutes en une variation dynamique de vitesse d'exécution des périphériques/composants en fonction de la charge courante de ceux-ci.

Plusieurs projets se sont intéressés à la gestion d'énergie dans un IaaS en proposant des politiques pour la plupart de ses équipements électroniques. Toutefois, dans le cas des serveurs, les politiques proposées sont davantage focalisées sur ses principaux composants : le CPU, le disque, les périphériques réseaux et la mémoire.

3.2 Gestion d'énergie sur quelques composants d'un serveur

L'état de l'art des politiques de gestion d'énergie que nous avons réalisé, a permis de définir le modèle de gestion d'énergie présenté en section 3.1. Chaque composant ayant des caractéristiques de performance et de consommation d'énergie différentes, il est généralement approprié d'avoir une stratégie distincte de gestion d'énergie pour chacun d'eux.
Pour chaque composant nous évoquons éventuellement ses caractéristiques matérielles particulières, puis nous présentons quelques exemples de politiques proposées.

3.2.1 Le CPU

Dans un IaaS, les performances d'exécution des VMs et le taux de consolidation de celles-ci sont liés aux caractéristiques architecturales du processeur les exécutant. L'architecture actuelle des processeurs permet d'atteindre des performances de calcul

assez élevées, ce néanmoins au prix d'une forte consommation énergétique de l'ordre d'une centaine de Watts [130]. Plusieurs travaux de recherches ont contribué à définir quelques politiques de gestion dynamique d'énergie.

Variation de vitesse de façon native

En règle générale, les politiques d'économie d'énergie applicable sur le CPU porte sur le changement d'état. Les approches proposées se composent de la mise du CPU en état de veille ou en état de vitesse donnée.
La mise en veille est traditionnellement privilégiée en période d'inactivité du CPU parce qu'elle abaisse la consommation énergétique du CPU au minimum. Pour davantage réduire l'énergie consommée en absence d'exécution, les nouvelles architectures de CPU possèdent un nouvel état nommé "*état de sommeil* (*sleep*)". Ce dernier se décline sur plusieurs niveaux progressifs : *sleep*,*deep sleep* et éventuellement *deeper sleep*. L'état de "sommeil" se caractérise par la perte des données du cache, une consommation énergétique nulle, mais un temps relativement (fonction de l'état courant) long pour revenir en état actif. Plus l'état de "sommeil" est profond, plus le temps de retour à l'état actif est long. Ainsi, en cas d'inactivité prolongée du CPU, celui-ci passe en état de veille et peut ensuite passer successivement par chacun de ces états de sommeil. Ce changement d'état permet ainsi de réduire la consommation énergétique du CPU.

Le passage du CPU en état de vitesse donnée pour réduire la consommation d'énergie du CPU s'effectue à travers les variations de fréquence d'exécution du CPU en fonction de leur charge. Autrefois, un CPU en exécution fonctionnait toujours à sa performance maximale indépendamment de sa charge. L'énergie qu'il consommait était gaspillée en cas de faible charge. Pour remédier à cette situation, les constructeurs d'équipements électroniques ont intégré une nouvelle technologie nommée *Dynamic Voltage and Frequency Scaling (DVFS)* dans les nouveaux processeurs.
Le DVFS est une technologie qui permet de modifier dynamiquement la fréquence (et la tension) d'un CPU en fonction de sa charge [154, 54, 77, 142]. Lorsque le CPU est faiblement chargé, sa fréquence (et sa tension) est graduellement abaissée de manière à ne point altérer la latence d'exécution du CPU. Le CPU fonctionne à sa fréquence (et sa tension) maximale (état actif) lorsqu'il est fortement sollicité ou possède une charge supérieure à un seuil défini (statiquement ou dynamiquement) [13, 49, 60].
L'économie d'énergie observable lors de l'utilisation du DVFS se base sur le rapport existant entre la tension et la fréquence. En effet, la puissance totale d'un CPU (P_t) peut être exprimée comme la somme de sa puissance dynamique (P_d) et sa puissance statique (P_s) :

$$P_t = P_d + P_s \tag{3.1}$$

La puissance dynamique est proportionnelle à la "capitance" C invariable, à la fréquence (f) et la tension (V) au carré [133] :

$$P_d = C * f * V^2 \tag{3.2}$$

La puissance statique représente la puissance au repos [18, 34]. Par conséquent, en réduisant la fréquence du CPU, ce qui conduit également à réduire la tension, la puissance du système est globalement réduite [123].

Regroupement des traitements

Cette section présente quelques approches basées sur le regroupement des traitements. Grâce au regroupement temporel, l'exécution des applications par le CPU peut être différée. C'est le cas de l'approche proposée par Elnozahy et al. et connue sous le nom de *Request batching* [32]. Le regroupement spatial utilise les possibilités de migration (la consolidation) d'applications entre systèmes [104] ou d'équilibrage de charge (load balancing).

Le "request batching" est un mécanisme qui consiste à regrouper les requêtes en mémoire pendant une période prédéfinie (nommée *batching timeout*) et à les traiter une fois le délai atteint. Avant l'expiration du délai, le CPU est inactif et est mis en état de veille. Le CPU possède dans cet état une consommation énergétique presque nulle. A l'expiration du délai, le CPU passe à l'état actif et fonctionne à performance maximale pour le traitement des requêtes accumulées.
Les serveurs étant utilisés à environ 30% de leur performance [82], cette configuration permet d'avoir de longues périodes d'inactivité et favorise ainsi la réduction de la consommation énergétique pendant chacune de ces périodes [59]. Le regroupement temporel des traitements permet de réduire au mieux la consommation énergétique du CPU.
Toutefois à l'opposé du DVFS, cette approche n'est avantageuse que si, pendant les périodes d'activités, le CPU possède une charge garantissant que la consommation énergétique du CPU à sa vitesse maximale est "proportionnelle" à sa charge. Afin de bénéficier des atouts du DVFS et du regroupement des traitements, une nouvelle approche les utilise indépendamment en fonction de la charge du système [10]. Ainsi, en absence de requêtes à traiter, le système est mis en état de veille jusqu'à ce qu'il soit réveillé par l'expiration du délai de regroupement (batching timeout). Grâce au DVFS, le CPU s'exécute successivement à différents niveaux de fréquence en fonction de sa charge courante. Utilisé de la sorte, le système réalise beaucoup plus d'économie d'énergie en garantissant une utilisation efficiente du CPU.

Le regroupement spatial des traitements se sert des atouts de migration d'applications qu'intègrent les OS [4]. Particulièrement, Srikantaiah et al. [125] exploitent les fonctionnalités de migration et de consolidation de VMs qu'apportent les technologies de virtualisation. Dans leur proposition, les VMs sont dynamiquement migrées entre les machines physiques en fonction de leurs caractéristiques, de leur charge courante et des performances de la machine physique. L'objectif idéal recherché par ces consolidations est de regrouper les VMs sur un nombre réduit de machines physiques pouvant les contenir. Ainsi, les machines non utilisées sont mises en état de veille ou

4. Notamment le système d'exploitation Linux

en état de "sommeil" ce qui favorise la baisse de l'énergie consommée par le système. Celles en cours d'utilisation, grâce au DVFS, fonctionnent à un niveau de fréquence correspondant à la charge courante de leurs CPU. En cas d'augmentation de charge d'un CPU, les machines physiques non exploitées sont successivement mises en état actif et les VMs sont migrées sur ces dernières afin d'offrir une bonne QoS.
Somme toute, les machines en cours d'exécution ne sont pas source de gaspillage d'énergie ; et l'arrêt des systèmes non utilisés permet d'accroître davantage la baisse de la consommation énergétique.
De même, le "load balancing" implémenté entre les CPUs d'un même ordinateur physique permet de regrouper les traitements et simuler ainsi une vitesse donnée pour l'ensemble des CPUs.

3.2.2 Le disque

La consommation énergétique des disques représente en moyenne 25 à 35% de la consommation totale d'un centre d'hébergement [62]. Fort de ce constat, de nombreuses approches ont été étudiées afin de réduire la consommation d'énergie des disques.

Variation de vitesse de façon native

Traditionnellement, les politiques de gestion d'énergie sont basées sur la définition d'une durée et d'un ensemble d'actions à effectuer en cas de dépassement de celle-çi. La valeur de cette durée est soit fixe, soit variable (en fonction des tranches horaires par exemple) soit adaptable (peut par exemple dépendre de la charge de travail).
L'une des premières techniques de gestion d'énergie pour disque présentée par Douglis et al. [30] adopte ce principe en définissant une durée d'inactivité. Cette approche consiste à faire transiter le disque en état de veille (*Spin down*) si cette durée d'inactivité prédéfinie est atteinte. La transition vers l'état actif d'un disque se produit dès l'arrivée d'une opération d'E/S. Cependant, ce réveil coûte généralement cher et amène une détérioration significative des performances du disque. Le compromis entre performance et économie d'énergie est donc, dans ce cas, étroitement lié à la valeur de la durée adoptée. Plus le disque passe en état de veille (parce que la valeur de la durée choisie est trop petite), plus le système sera pénalisé en terme de performance. Moins le disque passe en état de veille, moins l'économie d'énergie est possible. Le challenge de cette approche est le choix de la meilleure valeur de durée d'inactivité [78].

La transition vers l'état de veille ne survient qu'en cas d'inactivité prolongée du disque. Malheureusement, au regard du flux de données dans un IaaS, la période de latence entre 2 requêtes successives d'E/S ne permet pas fréquemment le passage en mode de veille des disques. De ce fait, les équipes de Gurumurthi et al. et de

Carrera et al. [16, 45, 44] proposent une nouvelle politique de gestion de la consommation énergétique des disques nommée *dynamic rotations per minute (DRPM)* par la première équipe et *multi-speed approach* par la seconde. Cette approche consiste à ajuster dynamiquement la vitesse de rotation du disque en fonction du volume des requêtes. En période d'inactivité, la vitesse de rotation est nulle (état de veille).

Regroupement des traitements

Le but de ces regroupements est une utilisation efficiente des disques. Cette efficience est liée au rapport entre le taux de requêtes traitées et la consommation énergétique du périphérique. Grâce au regroupement temporel, les demandes de E/S sur disque sont regroupées et subissent un traitement différé. Grâce au regroupement spatial, les données sont regroupées sur des disques en fonction de leur fréquence d'accès ainsi que de leurs données voisines (en terme d'adressage).

Le regroupement temporel des traitements distingue ces 2 types d'opérations : la lecture et l'écriture sur disque.
Pour la lecture, Weissel et al. propose de regrouper et de reporter le traitement des demandes de lecture autant que les applications présentent la flexibilité requise pour le faire [148]. A cet effet, Weissel introduit dans les OS une interface logicielle utilisée par les applications pour spécifier un délai maximal d'attente des demandes. Cette interface permet aussi d'activer un drapeau d'annulation pour des demandes d'E/S. Ainsi, quand le disque est en état de veille, l'OS peut (i) soit annuler la demande de lecture si le drapeau d'annulation a été activée par l'application ; (ii) soit différer le traitement des demandes de lecture jusqu'à expiration du délai d'attente.
Le principe est pratiquement identique quand il s'agit des demandes d'écriture sur disque. Les blocs de données modifiées sont conservées en mémoire cache tant que le disque est en état de veille. D'un côté, ces données sont directement sauvegardées sur disque lorsque ces derniers deviennent actifs (*write-back with eager update (WBEU)*). D'un autre côté, les données modifiées sont temporairement sauvegardées dans un fichier log en mémoire (ou sur n'importe quel périphérique de stockage persistant) pendant toute la période de veille du disque. Elles sont ensuite sauvegardées sur disque au moment où celui-ci devient de nouveau actif (*write-through with deferred update (WTDU)*) [156].
Dans chacun des cas de regroupement, les demandes sont traitées par lot, ce qui augmente les périodes d'inactivité des disques. Pendant ces phases d'inactivité, les opérations de gestion d'énergie applicables de façon native sur le disque permettent de réduire sa consommation énergétique.

L'exploitation du regroupement spatial permet de regrouper les données sur les disques en fonction de leur taux d'utilisation. Colarelli et al. [23] définit une nouvelle technologie de configuration de disque nommée *MAID(Massive Array of Idle Disks)* et propose d'organiser les disques en "disque de données (DD)" et "disque cache (DC)". Les DD servent à la sauvegarde des données. Les DC servent à la réplication des données récemment utilisées et de celles qui leur sont contiguës. Les DC sont

utilisées en permanence parce que toute opération de lecture y est exécutée. De plus, les DC servent de mémoire tampon pour les opérations d'écritures. Les données modifiées sont directement sauvegardées sur les DC[5]. L'écriture sur le DD est différée et réalisée lors de son passage en état actif[6] [118, 36]. Ainsi, pendant une durée relativement longue, il est possible d'appliquer les politiques de gestion d'énergie ci-dessus citées sur les DD et réaliser de ce fait une réduction de la consommation énergétique des disques.
Cette idée a été étendue aux architectures *Redundant Array of Independent (or inexpensive) Disks (RAID)* [111]. Verma et al. [138] propose *sample-replicate-consolidate mapping (SRCMap)*, une approche basée sur la consolidation des données par réplication/copie entre disques. SRCMap représente une technique d'optimisation de l'utilisation des disques d'un cluster en copiant uniquement les *données actives* (donnée la plus souvent utilisée). Pour chaque disque, SRCMap identifie les blocs de données actives qu'il copie sur d'autres disques. La petite taille des copies favorise la création de multiples copies sur un ou plusieurs disques ou inversement permet à un disque d'accueillir les copies de plusieurs disques. SRCMap s'assure de regrouper l'ensemble des données et les répliques sur un nombre minimum de disques. Les disques non utilisés sont de ce fait mis en état de veille et toute requête d'E/S est dirigée vers l'un des disques actifs contenant la copie de la donnée souhaitée. La réduction du nombre de disques actifs permet de baisser la consommation énergétique des disques. Les opérations d'écriture sont différées et des espaces sont prévus sur chaque disque et sont utilisés comme mémoire tampon[7] [95].

3.2.3 Le périphérique réseau

Des mesures d'équipements réseaux d'entreprises révèlent une utilisation de moins de 30% [124, 58]. Ils sont donc pour la plupart du temps inactifs mais leur consommation énergétique reste aussi importante. Les techniques de gestion d'énergie appliquées à ces équipements (notamment à une carte réseau) ont pour but de réduire l'énergie gaspillée lorsque le réseau est inactif.

Variation de vitesse de façon native

La première approche de gestion d'énergie de carte réseau tire avantage des longues périodes d'inactivité régulièrement observées dans le fonctionnement d'une carte réseau. Ainsi, pendant les périodes d'inactivité, la carte réseau est mise en état de veille. Nedevschi et al. [99] a effectué un travail initial d'exploration portant sur la réduction d'énergie des réseaux sans dégradation de performance de celui-ci. Par analogie au CPU, la carte réseau intègre un support matériel permettant le changement d'état. En l'absence totale de trafic, la carte réseau est mise en état de veille. Sa

5. *Write-through* policy
6. *Write-back* policy
7. Write off-loading (WO)

consommation énergétique est réduite au minimum ou est presque nulle. Toutefois, en présence d'un trafic réseau (faible ou dense), la carte réseau fonctionne au mieux de ses performances et consomme ainsi la même quantité d'énergie ; ce qui conduit à nouveau à un gaspillage d'énergie.

Pour remédier à cette situation, Nordman et Christensen [100] ont introduit le concept d'*Adaptive Link rate (ALR)* comme un moyen de changer dynamiquement le débit de traitement d'un réseau full-duplex en fonction de la charge de trafic qu'il traite. Sur ce principe, plusieurs approches ont été élaborées [42, 40, 41, 69]. Notamment, Gunaratne et al. [41] a implémenté une approche nommée *Adaptive Link rate (ALR)*. Le fonctionnement d'une carte réseau à faible débit étant plus économique (sur le plan de l'énergie) qu'une exécution à haut débit, l'approche ALR consiste à faire varier dynamiquement le débit d'exécution de la carte réseau en fonction de la charge du trafic reçu. Ce contrôle et cette variation de débit réduit ainsi la consommation énergétique globale de la carte réseau. Bien que la technique ALR soit efficace en terme d'économie d'énergie quelle que soit la charge du trafic, l'économie réalisée est inférieure au passage de la carte réseau en état de veille. Ananthanarayanan et al.[2] a proposé une approche d'économie d'énergie utilisant conjointement le passage en état de veille et l'ALR qu'il a appliqué sur les switchs d'un réseau local.

Regroupement des traitements

Pour la mise en place du regroupement temporel, des travaux de recherches se sont également intéressés à faire du buffering au niveau des périphériques réseaux. Notamment, Klausmeier et al. [64] a proposé un procédé et un appareil pour mettre en tampon des informations dans un réseau numérique.
Afin d'utiliser le regroupement spatial, la technique de *NIC teaming* [88](généralement traduit par *couplage de carte réseaux* en français) a été largement adopté. Le NIC teaming consiste à faire fonctionner 2 ou plusieurs cartes réseaux ensemble afin d'améliorer la bande passante du serveur et de fournir une redondance réseau. Wei Liu et al. [84] a ainsi proposé une solution d'économie d'énergie en se basant sur ce principe. Dans leur approche, toute carte réseau du couplage est capable de recevoir du trafic réseau. Chaque contrôleur de carte réseau est également adapté et configuré pour recevoir des informations associées à la charge du trafic. Cette configuration lui permet également de notifier les cartes réseaux en état de veille afin de les rendre active. Ainsi, en absence de trafic ou quand la charge du trafic est inférieure à un seuil prédéfini, un sous ensemble de cartes réseaux sont mises en état de veille. Elles sont successivement mises en état actif en fonction du trafic reçu et ce dernier est équitablement distribué entre les cartes réseaux actives.

3.2.4 La mémoire

La gestion de la consommation énergétique de la mémoire revêt actuellement une très grande importance et fait l'objet de plusieurs travaux de recherches [17, 27, 61, 155, 107].

Variation de vitesse de façon native

Le modèle de gestion d'énergie de la mémoire le plus répandu suit les spécifications de Rambus Dynamic Random Access Memory (RDRAM) [56, 72, 47]. Il est basé sur le principe d'une mémoire capable de fonctionner dans de multiples niveaux de puissance. Dans un système de mémoire RDRAM, chaque mémoire peut être réglée de façon indépendante à une puissance apparente appropriée à un état : *actif*, *veille*, *nap*, ou *powerdown* [26].
Dans l'état actif, toutes les parties de la mémoire sont actives. Il représente le seul état dans lequel les opérations de lecture et d'écriture sont réalisables. L'état actif constitue également le plus consommateur en énergie. Dans les autres états, l'ensemble des données est préservé et seules certaines parties du circuit mémoire sont activées [122]. Ces états sont appelés *états de faible consommation*. La consommation énergétique de la mémoire décroit au fur et à mesure qu'elle est passée à l'état de faible consommation inférieur. Toutefois, le passage d'un état de faible consommation à l'état actif est généralement coûteux en terme d'énergie et de délai de réponse [80].
La mémoire passe en état de veille au terme de toute exécution de requête si la file d'attente de l'ordonnanceur est vide. Elle peut également passer d'un état de faible consommation à un autre suite à l'expiration d'un seuil d'inactivité. Le retour à l'état actif survient à l'arrivée d'une requête à traiter [74, 79].

Les travaux antérieurs ont essentiellement porté sur l'utilisation des états de faible consommation pendant les périodes d'inactivité entre les requêtes de mémoire pour économiser l'énergie. Avec l'avènement des nouvelles mémoires de type DDR (Double Data Rate) et la possibilité qu'elles ont de fonctionner à des vitesses différentes en fonction de leur charge courante ; le *DVFS mémoire* a été adopté par plusieurs projets afin de réduire la consommation énergétique des mémoires [25, 28, 70, 87].
Notamment, Deng et al. [29] propose une approche nommée *MemScale* basée sur le changement dynamique de fréquence de la mémoire. Pratiquement, MemScale met en oeuvre la variation dynamique de fréquence et de tension du contrôleur mémoire en fonction des données relatives à l'état du système (taille de la file d'attente de l'ordonnanceur, nombre total de défaut de cache[8]) que lui retourne l'algorithme de monitoring qu'il intègre. A chaque quantum, MemScacle monitore le système et sélectionne une fréquence qui minimise l'énergie globale du système mémoire et la dégradation de performance des applications. La mémoire fonctionne ainsi à basse

8. Total LLC (Last-level Cache) Misses (TLM)

fréquence quand elle est sous chargée et au meilleur de ses performances dans le cas contraire.

Regroupement des traitements

Les approches précédentes, appliquées sur les barrettes mémoire de façon individuelle, permettent d'économiser de l'énergie en effectuant des transitions entre l'état actif et les états de faible consommation. Toutefois, au regard du coût (énergie et délai d'exécution) qu'engendrent ces transactions, ces approches ne sont véritablement efficaces que dans le cas d'une période d'inactivité suffisamment longue [33, 53].
Cependant, les applications consommatrices de mémoire ne font généralement face qu'à de très courtes périodes d'inactivité. Par conséquent, le mécanisme de transition d'état, en vue de réduire la consommation énergétique, nécessite l'exploitation du regroupement spatial des traitements. Ce dernier concerne le groupement de pages permettant de disposer de plus longues périodes d'inactivité [128, 83, 81].
Wu et al. [150] proposent un nouveau modèle de mémoire nommée *RAMZzz* intégrant les techniques de gestion d'énergie. RAMZzz effectue la migration des pages en fonction des différentes fréquences d'accès à celles-ci. RAMZzz définit 2 catégories de mémoire : *hot*[9] et *cold*[10] et regroupe les pages sur l'une ou l'autre catégorie de mémoire. La première (hot) contient les données les plus souvent consultées et possède de très courtes périodes d'inactivité. La seconde (cold) contient les pages les moins consultées et a de ce fait, des périodes relativement longues d'inactivité. Ces périodes sont utilisées pour les transitions d'état en vue d'économiser la consommation énergétique. En cours de fonctionnement, les pages sont dynamiquement placées sur les mémoires en fonction de leur taux d'utilisation. Ce regroupement permet de consolider les petites périodes d'inactivité en longues périodes et d'appliquer ainsi les politiques de transaction d'état. D'un autre côté, les mémoires "hot" sont exécutées à une performance donnée en fonction de leur charge ; ce qui favorise également l'économie d'énergie.

Le regroupement temporel dans le cas de la mémoire est étroitement lié au regroupement temporel du CPU. En effet, pendant que le CPU est inactif, la file d'attente de l'ordonnanceur étant vide la mémoire est également mise en veille.

3.2.5 Gestion d'énergie d'autres composants dans un IaaS

A un niveau de gestion plus macroscopique, les opérations de variation de vitesse de façon native ou par émulation sont également applicables à un serveur physique en tant qu'entité [89, 35], à tout autre équipement réseaux (swicth ou routeur) [43, 121, 110] ou aux autres équipements utiles pour le refroidissement/ventilation d'un centre d'hébergement [37].

9. Chaud
10. Froid

3.3 Objectifs des politiques existantes

Au regard du modèle de gestion d'énergie défini en section 3.1 et des approches présentées en section 3.2, l'ensemble des solutions de gestion d'énergie consiste entièrement à des opérations de changement de vitesse d'exécution. Ces variations sont exécutées de façon native pour certains périphériques ou sont émulées par des regroupements de traitements.

Le but idéalement recherché par ces approches de gestion d'énergie est de respecter le SLA tout en consommant le moins possible d'énergie. Cet idéal est atteint en état actif ou veille. En effet, un périphérique en état actif dispose d'une charge (définie par le constructeur) lui permettant de garantir le SLA souscrit par les clients. Dans cet état, sa consommation énergétique est optimale. En état de veille, la consommation énergétique est quasi nulle et la notion de respect du SLA n'est pas considéré.

Le challenge des approches de gestion d'énergie concerne l'état de vitesse donnée. Dans cet état, la vitesse d'exécution du périphérique/composant est ajustée en fonction de la fluctuation de sa charge. La baisse de la vitesse d'exécution du périphérique/composant a pour effet d'accroître le temps de traitement des requêtes. Cette croissance conduit corollairement à une altération de la QoS et au non respect du SLA. La hausse de la vitesse d'exécution du périphérique/composant permet de garantir le SLA souscrit mais peut conduire à un gaspillage énergétique. Il serait donc souhaitable d'obtenir une proportionnalité "parfaite" entre la vitesse d'exécution des périphériques/composants et leur consommation énergétique.

3.4 Synthèse

Dans cette section, nous avons présenté un modèle de gestion d'énergie ainsi que quelques implémentations pour certains périphériques (CPU, disque, carte réseau et mémoire) d'un serveur.
Le modèle de gestion d'énergie défini met en exergue le fait que les solutions actuelles de gestion d'énergie sont basées sur le changement d'état d'exécution des périphériques. Ce changement d'état peut être mis en œuvre de façon native via des fonctionnalités intrinsèques à certains périphériques. Et peut également être mis en œuvre par simulation via des regroupements temporel ou spatial des traitements qu'effectue le périphérique. Afin de valider ce modèle, nous avons présenté un état de l'art succinct de quelques politiques de gestion d'énergie applicables sur le CPU, le disque, la carte réseau ou la mémoire.

L'analyse des politiques existantes a relevé le fait qu'en plus de devoir respecter le SLA contracté avec ses clients, la problématique de gestion équitable de l'énergie

consommée par l'infrastructure constitue une préoccupation importante d'un fournisseur. Dans le cadre de cette thèse, nous nous intéressons à cette problématique en nous focalisant sur la gestion équitable du processeur des systèmes virtualisés dans un IaaS. Au delà du caractère d'équité recherché dans cette gestion, notre objectif est d'offrir des mécanismes permettant de minimiser l'énergie consommée par ces processeurs tout en respectant le SLA.

Dans le chapitre suivant, nous examinons les motivations de nos travaux de recherche et nous montrons les insuffisances des solutions actuelles de gestion d'énergie des processeurs en environnement virtualisé.

Deuxième partie

Problématique et État de l'art

Chapitre 4

Problématique

Contents

Dans le chapitre précédent, nous avons fait état de quelques approches de gestion d'énergie conformément à un modèle de gestion d'énergie que nous avons préalablement défini. Les solutions étudiées consistent globalement en des changements de vitesse d'exécution des périphériques. Ces changements sont réalisés via des fonctions natives fournies par le périphérique ou via des regroupements (spatiaux ou temporels) de traitements. Ces derniers ayant pour objectif final de simuler l'état de vitesse donnée pour des périphériques ne disposant pas de cette fonctionnalité.

Une analyse de ces solutions de gestion d'énergie montre qu'elles peuvent entrainer une violation du SLA des services. L'adoption de ces mécanismes de réduction de la consommation énergétique dans un IaaS devrait en conséquence se faire de manière intelligente de façon à limiter son impact sur le SLA. Le présent chapitre soulève ainsi la problématique liée à l'atteinte de ces deux objectifs essentiels inhérent de la gestion de ressource dans un IaaS : le respect du SLA et la gestion optimale des ressources et en particulier l'énergie consommée.

Dans ce chapitre, nous présentons le problème généralement observé dans un IaaS face aux 2 objectifs sus-cités. Cette présentation sera essentiellement basée sur les principes du modèle de gestion d'énergie défini en section 3.1. Ensuite, nous illustrons cette problématique en nous focalisant sur la gestion du CPU dans un système virtualisé.

4.1 Problématique générale

Dans cette section, nous rappelons d'une part les exigences d'un système de gestion de ressources dans un IaaS et les 2 niveaux de gestion qui le caractérise. D'autre part, nous relevons les limites des politiques de gestion d'énergie actuelles quant à l'atteinte de ces exigences.

4.1.1 Exigences d'un système de gestion de ressources dans un IaaS

Lors de la souscription d'un service dans un IaaS, le client effectue une réservation (immédiate ou programmée) dans laquelle il spécifie les caractéristiques matérielles du service désiré. Il définit en accord avec le fournisseur un SLA à garantir ainsi que l'ensemble des pénalités liées au non respect de celui-ci.

Au déclenchement de la réservation, les VMs correspondant au besoin du client lui sont allouées. Tout au long de l'exécution de ces VMs, le gestionnaire de ressources a la charge de veiller continuellement à ce que le SLA soit respecté. Notamment, avant d'effectuer toute opération de migration, le gestionnaire de ressources doit s'assurer qu'au terme de l'opération le SLA lié à la VM est toujours respecté. En somme, d'un point de vue utilisateur, quels que soient les changements architecturaux auxquels est soumis l'infrastructure matérielle, le gestionnaire de ressources doit garantir le SLA exigé.

D'un point de vue fournisseur, la gestion des ressources suppose la mise en place de politiques liées à l'utilisation efficiente de l'ensemble des ressources. Il s'agit de s'assurer que la consommation énergétique de l'infrastructure est le plus possible proportionnelle à la charge de travail qu'elle reçoit et ceci sans dégradation du SLA.

En adéquation à ces 2 niveaux d'exigences, la gestion de VMs dans un IaaS s'effectue généralement suivant 2 échelles. Dans la section suivante, nous présentons un modèle général de gestion de VMs conforme à cette subdivision.

4.1.2 Modèle général de gestion de VMs

La gestion des VMs dans un IaaS s'effectue de 2 manières : de manière *distribuée* par l'équilibrage de charge entre les différents ordinateurs (communément appelés *nœuds*) de l'infrastructure et de manière *centralisée* par l'ordonnanceur de VMs implanté dans chaque nœud :

1. ***Gestion distribuée*** : La première échelle (le service d'équilibrage de charge) est réalisée à travers les opérations de migration et de consolidation. Dès le démarrage et tout au long de l'exécution d'une VM, le service d'équilibrage de charge choisit le nœud sur lequel placer la VM. Ce service possède une vue d'ensemble de l'infrastructure du IaaS. Ainsi selon l'état de cette dernière, il migre les VMs pour un usage optimal de l'infrastructure.
   L'idéal pour un usage optimal de l'infrastructure serait de pouvoir réaliser une "*consolidation parfaite*"[1] des VMs. Cette dernière devrait permettre de rassembler toutes les VMs en cours d'exécution sur un ensemble réduit de nœuds. Ces migrations permettraient d'obtenir : (i) d'un côté des nœuds globalement chargés de manière idéale et fonctionnant au meilleur de leur performance et (ii) de l'autre côté, des nœuds non utilisés mis en état de veille[2] afin de réduire la consommation énergétique de l'infrastructure.

   Toutefois, la mise en place d'une "consolidation parfaite" est rarement possible du fait de la taille limitée de la mémoire du système hôte [101, 57]. En effet, toute VM a besoin de mémoire physique minimale (fixée par l'administrateur ou exigée par l'hyperviseur) pour son fonctionnement. De ce fait, le nombre de VMs capable de s'exécuter en même temps sur un nœud est limité par la taille de la RAM de celui-ci.

   Face à cette restriction, la gestion distribuée (à elle seule) des VMs ne garantit pas le respect d'une des exigence des gestionnaires de ressource dans un IaaS : l'utilisation optimale de l'infrastructure. Dans ce contexte, l'adoption des mécanismes de gestion centralisée de VMs intervient généralement en complément à la gestion distribuée [97, 96, 71].

2. ***Gestion centralisée*** : La deuxième échelle de gestion de VMs se situe au sein du nœud et est exécutée par l'ordonnanceur (*scheduler*) de VM. L'ordonnanceur de VM désigne le composant de la VMM auquel est assigné, entre autres, 3 rôles utiles au bon fonctionnement des VMs : (1) allouer la ressource aux VMs, (2) veiller au respect de la capacité de ressource souscrit pour la VM et (3) choisir l'ordre d'exécution des VMs sur le système hôte. L'ordonnanceur de VMs effectue ces tâches avec pour objectif d'effectuer un partage équitable

1. Dans ce document, nous appelons "*consolidation parfaite*", toute situation dans laquelle tous les nœuds exécutant des VMs ont une charge CPU supérieure à un seuil (de surcharge) prédéfini.
2. Dans le cas d'un nœud, la mise en veille représente l'un des états C-states (C1-C3) définis par la norme ACPI

des ressources entre les VMs qu'il gère.

Dans un système virtualisé dans lequel une politique de gestion d'énergie est définie [3], cette dernière ajuste dynamiquement la vitesse d'exécution de la ressource en fonction de sa charge globale. De cette manière, à la suite des opérations de consolidation réalisées par le service d'équilibrage de charge, la définition des politiques de gestion permet à chaque nœud de fonctionner à un niveau de performance relatif à la charge courante de chacun de ses périphériques. Grâce à l'utilisation conjointe de ces 2 niveaux de gestion de VMs, il est possible de maitriser la consommation d'énergie des nœuds.

Toutefois, l'ordonnanceur de VMs sélectionnant et exécutant les VMs indépendamment de la vitesse d'exécution de ses périphériques, une réduction de cette vitesse influe directement sur la performance des VMs. De ce fait, le respect du SLA requiert une coordination entre la gestion centralisée des VMs et la politique de gestion d'énergie mise en place.

L'objectif de cette thèse est de contribuer à la mise en place de mécanisme de gestion de ressources dans un environnement virtualisé respectant, au mieux, ces 2 exigences. Dans le cadre de nos travaux de recherche, nous nous concentrons uniquement sur l'aspect gestion centralisée des VMs.

4.1.3 Limites des approches actuelles

Les limites des approches actuelles de gestion de ressources s'observent principalement en environnement virtualisé [4]. Cette limitation se caractérise par la difficulté à assurer à la fois la réduction de la consommation énergétique des infrastructures et le respect du SLA.

De la réduction de la consommation énergétique

Le modèle de gestion d'énergie que nous avons défini en section 3.1.1 a permis d'observer que l'ensemble des périphériques présents dans un ordinateur ont la particularité d'être des *périphériques à vitesse variable*. Un périphérique est soit en veille (*vitesse presque nulle*) ou s'exécute à une vitesse quelconque (*vitesse maximale ou vitesse donnée différente de la vitesse maximale*).

L'analyse effectuée en section 3.3 sur les politiques de gestion d'énergie existantes nous montre que les mécanismes actuels de réduction de la consommation énergé-

3. La définition d'une politique de gestion d'énergie suppose que le périphérique (ressource) est sujet au changement de vitesse tel que présenté en section 3.1.1

4. Environnement qui constitue notre domaine de recherche

tique des périphériques reposent sur cette particularité. En fonction de la charge du périphérique, la politique de gestion d'énergie adoptée consiste soit à le mettre en veille soit à changer la vitesse à laquelle il s'exécute. Cette variation de vitesse n'étant pas toujours disponible pour tous types de périphériques, le regroupement (spatial ou temporel) des traitements à effectuer par le périphérique est utilisé comme levier pour simuler ce changement de vitesse d'exécution. Ceci permet d'optimiser la consommation énergétique locale au périphérique et celle de l'infrastructure à une échelle plus grande.

Toutefois, le changement de vitesse d'exécution d'un périphérique influe sur la vitesse d'accès/utilisation dudit périphérique. Indirectement, cette influence détériore le SLA des applications qui l'utilise.

Du respect du SLA

Le respect du SLA d'un client consiste à satisfaire son besoin en ressources. Au sein d'un IaaS, le service souscrit par les clients s'exprime sous forme de capacité. Cette capacité, identifiée par un ensemble de VMs, représente une quantité [5] de ressources (CPU, espace disque, RAM, NIC) d'une machine physique. Au cours du fonctionnement de ses VMs, le client souhaite disposer de la totalité de cette capacité de ressources s'il en a besoin. Cette garantie de disponibilité réfère au respect du SLA.

Supposons un nœud virtualisé sur lequel sont déployées N VMs de capacité respectives α_1, α_2,..., α_N, définies sur la base d'un périphérique s'exécutant à sa vitesse maximale V_{max}. Tout au long de l'exécution des VMs, chacune génère une charge L_i (i=1,2,...,N) telle que la charge totale de la ressource utilisée est $L_{tot} = \sum_{i=1}^{N} L_i$.
En règle générale, si une politique de gestion d'énergie est adoptée pour le périphérique concerné, sa vitesse d'exécution à un instant t varie en fonction de sa charge totale c'est-à-dire : $V_t = f(L_{tot})$. Les VMs étant migrées d'un nœud à un autre, le gestionnaire de ressources se doit d'allouer une part de ressource β_i, avec

$$\beta_i = \alpha_i \times \frac{V_t}{V_{max}} \tag{4.1}$$

à chaque VM afin de satisfaire la capacité souscrit α_i. Le gestionnaire n'étant pas conscient du changement de vitesse du périphérique, lors de ce calcul, la vitesse courante est toujours égale à la vitesse maximale $V_t = V_{max} \Rightarrow \alpha_i = \beta_i$.
Cependant :

1. Si le périphérique est sous chargé, la politique de gestion d'énergie abaissera sa vitesse d'exécution à une vitesse V_t pouvant absorber sa charge totale L_{tot}.

5. Une part de ressources

Dans ce contexte, nous aurons : $V_t < V_{max} \Rightarrow \beta_i < \alpha_i$. Les VMs n'obtiendront pas la capacité souscrite.

Cette situation est d'autant plus pénalisante pour les VMs surchargées. En considérant par exemple notre scénario ci dessus, supposons que VMLoad représente la consommation (réelle) de la ressource vue par la VM. Cette charge représente une part de la charge totale de la ressource du point de vue du nœud physique c'est-à-dire : $L_{tot} = \sum_{i=1}^{N} \alpha_i \times \text{VMLoad}_i$
Si la capacité α_i d'une VM est très inférieure à la capacité totale de la ressource ($\alpha_i << 100$), l'apport de la charge L_i à la charge totale L_{tot} est négligeable. Supposons une VM_j utilisant 100% ($\text{VMLoad}_j = 100$) de sa capacité α_j ($L_j = \alpha_j$) avec $\alpha_j << 100$ (*c'est-à-dire avec une capacité très inférieure à la capacité totale*). Si l'ensemble des N-1 autres VMs ont chacune une charge presque nulle ($L_i \simeq 0$ pour i=1,..,N,$i \neq j$), la charge totale de la ressource sera sensiblement égale à la charge de la VM_j ($L_{tot} = L_j$).

La charge totale L_{tot} de la ressource donnera ainsi une vision "faussée" de l'utilisation de la ressource. Cette faible utilisation "fictive" de la ressource (*et de là la baisse de sa vitesse d'exécution bien qu'il existe une VM surchargée qui en a besoin*) aura un impact sur la performance des VMs.
La performance des N-1 VMs sous chargées ne sera pas "réellement" impactée puisqu'elles n'utilisent pas intensément la ressource. Ce qui n'est malheureusement pas le cas de la VM_j. Cette dernière sera victime de cette baisse de vitesse et subira une réelle dégradation de sa performance. Cette dégradation s'explique par le fait que la VM_j surchargée, ne reçoit pas en totalité la capacité de ressources pour laquelle elle a souscrit à cause des politiques d'économie d'énergie appliquées.
Ce dysfonctionnement est regrettablement observable pour l'ensemble des ressources utilisant de façon "naïve" les politiques de gestion d'énergie actuelles.

En somme, cette stratégie de gestion d'énergie engendre un non respect de SLA dû au fait que le gestionnaire de ressources n'est pas conscient du changement de vitesse d'exécution de la ressource.

2. Dans le cas où aucune politique de gestion d'énergie n'est adoptée afin d'éviter ces effets de bords, la vitesse d'exécution du périphérique demeure maximale. Ainsi quelle que soit la charge totale L_{tot} de la ressource, chaque VM recevra toujours la capacité de ressource souscrite ($V_t = V_{max} \Rightarrow \alpha_i = \beta_i$). Cette stratégie a l'avantage de garantir le respect du SLA.
Toutefois, pour un périphérique dont la charge totale (L_{tot}) est très faible ($L_{tot} \simeq 0$), le maintien de la vitesse maximale est une réelle source de gaspillage énergétique.

4.1.4 Synthèse

Dans cette section, nous avons relevé 2 contraintes importantes de tout système de gestion de ressources : le respect du SLA et la réduction de la consommation énergétique de l'infrastructure. Nous avons également observé qu'il n'est idéalement pas possible de respecter ces 2 contraintes en utilisant les approches de gestion de ressources classiques. Particulièrement, dans le cas de systèmes virtualisés, nous avons observé que l'état d'utilisation d'un périphérique/composant peut ne pas toujours refléter le besoin réel des VMs qui l'utilisent. Cette vision "faussée" de l'usage qui est fait d'un périphérique/composant peut entrainer une dégradation de performance beaucoup plus considérable pour les VMs surchargées.

La section suivante permet de circonscrire le périmètre de nos travaux de recherches.

4.2 Position du problème : cas particulier du CPU

Face à la problématique générale portant sur le respect du SLA et la réduction de la consommation énergétique des périphériques/composants dans un système virtualisé, nous nous attardons particulièrement au cas du CPU.
Le SLA est normalement assuré par le gestionnaire de ressources. Le DVFS, tel qu'implanté à l'origine par les constructeurs ou amélioré par divers travaux de recherche, est l'approche la plus répandue en terme de politique de gestion d'énergie d'un CPU.
Dans cette section, nous montrons comment la mise en place du DVFS, sans aucune précaution préalable, notamment la prise en compte des spécificités d'un système virtualisé, illustre bien le problème exposé en section 4.1.3. Dans la section suivante, nous présentons le fonctionnement général d'un ordonnanceur de VM (gestionnaire de CPU). Ensuite, nous présentons le fonctionnement général du DVFS comme politique de gestion d'énergie. Enfin, nous illustrons, via des scénarios, le problème de coordination entre ces 2 éléments .

4.2.1 Ordonnanceur de VMs : principe de fonctionnement

Le fonctionnement des ordonnanceurs de VMs (regroupés sous le nom de *proportional-share scheduler* dans la littérature) vise à affecter une fraction de temps du processeur du système hôte à une VM en se basant communément sur le paramètre *capacité* descriptif d'une VM [20] et d'une politique d'allocation.
La ***capacité*** d'une VM représente la fraction maximale de temps CPU pour laquelle

la souscription a été faite[6].
La ***politique d'allocation*** du temps processeur aux VMs permet de subdiviser les ordonnanceurs en 2 catégories : les ordonnanceurs à *capacité fixe* et ceux à *capacité variable*.
Dans un ***ordonnancement à capacité fixe***, la capacité de CPU souscrit pour chaque VM est toujours garantie. Ceci suppose que chacune des VMs obtient invariablement ses tranches de temps processeur qu'elle en ai besoin ou pas. En cas de surcharge d'une VM, elle ne peut aucunement bénéficier des tranches de temps processeur éventuellement non utilisées par d'autres VMs. Ce type d'ordonnanceur fait partie de la famille des ordonnanceurs nommée *non-work conserving*.
Dans un ***ordonnancement à capacité variable***, la capacité de CPU de chaque VM est garantie si nécessaire. Dans le cas contraire, les tranches de temps processeur non utilisées sont redistribuées entre les VMs actives qui en ont besoin. Ceci signifie que les processeurs d'un ordinateur utilisant un ordonnanceur à capacité variable ne deviennent inactifs qu'en absence de VM à exécuter. Ce type d'ordonnanceur fait partie de la famille des ordonnanceurs nommée *work conserving*.

Pour illustrer le fonctionnement de ces 2 familles d'ordonnanceurs, supposons un nœud sur lequel sont exécutées 2 VMs de même capacité CPU (par exemple = 50%). Si l'ordonnanceur est de type *capacité fixe*, chaque VM recevra exactement 50% de tranches de temps processeur si besoin ou pas. Dans le cas d'un ordonnanceur à *capacité variable*, si l'une des VMs devient inactive, la VM active peut recevoir jusqu'à 100% du processeur si elle est en situation de forte charge de travail.

Dans un IaaS, l'ambition du fournisseur est de satisfaire les demandes de tous ses clients tout en réduisant au mieux ses dépenses afin d'assurer un gain financier. Pour cette raison, il utilisera dans la plupart des cas un ordonnanceur à capacité fixe.

4.2.2 DVFS : principe de fonctionnement

De nos jours, la majorité des processeurs intègrent la technologie *DVFS*. Cette technologie se sert principalement des spécifications d'état du processeur définies par la norme *ACPI*[7](*interface avancée de configuration et de gestion de l'énergie*)[8].

Approche générale La norme ACPI vise à uniformiser et à améliorer l'accès et l'utilisation des périphériques matériels d'un ordinateur. Dans cet objectif, elle offre aux OS une interface standardisée permettant de communiquer avec les différents périphériques matériels d'un ordinateur via des signaux. La norme ACPI décrit de

6. Cette capacité est déclinée en valeurs minimale et maximale pour certaines technologies de virtualisation
7. Advanced Configuration and Power Interface - ACPI
8. Norme développée conjointement par HP, Intel, Microsoft, Phoenix Technologies et Toshiba

la même manière une interface pour la configuration, la gestion de la consommation électrique et la surveillance (monitoring) des périphériques. Ces spécifications ont l'avantage de permettre l'implémentation rapide et simplifiée de nouvelles fonctionnalités de gestion d'énergie et par ricochet d'infrastructure soucieuse de l'énergie qu'elle consomme.
La gestion d'énergie est effectuée grâce aux états de *fonctionnement* (*C-states*) et de *performance* (*P-states*) de l'ordinateur et des périphériques que définissent la norme ACPI[9]. Les états C-states se déclinent en sous états *C0-C3* indiquant respectivement : l'état de marche, d'arrêt, d'arrêt d'horloge et de veille du processeur. Le traitement des instructions reçues par le processeur se fait en état C0. Les autres états servent à la baisse de la consommation électrique du processeur, ceci au prix d'un retour à l'état C0 de plus en plus long.
En outre, en état de marche (état C0), le processeur peut être dans l'un des états de performance (P-states déclinés en sous états *P0-Pn*[10]) existants sur la plateforme concernée. Chacun des P-states désigne une combinaison entre la fréquence et la tension du processeur (propriété exploitée par le DVFS). Bien que dépendant de l'implémentation, l'état P0 (des états P-states) est toujours l'état de performance le plus élevé et les états P1 à Pn sont successivement les états de performance inférieurs.

Ainsi, pour la mise en œuvre du DVFS, l'OS exploite l'interface logicielle standardisée mis à sa disposition (par la norme ACPI) pour ajuster dynamiquement la fréquence d'exécution du processeur. En effet, en état de marche (état C0) et en fonction de sa charge courante, l'état de performance du CPU passe éventuellement de l'état P0 à Pn et inversement.

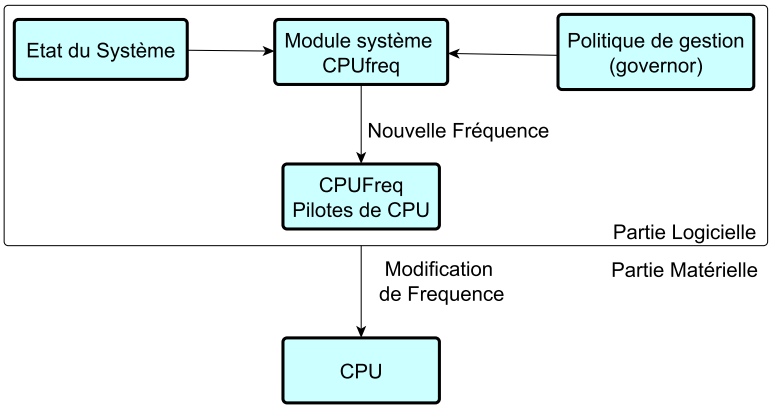

Fig 4.1. Principe de fonctionnement de Cpufreq

DVFS sous Linux Sous linux, l'interface appelée *cpufreq* (Figure 4.1) et située à l'intérieur du noyau fournit un ensemble d'instructions de contrôle de la fréquence du processeur [106]. Ce contrôle de fréquence est sujet à différentes politiques de

9. La prise en compte de ces états requière une activation préalable dans le BIOS au démarrage du système
10. Dans une limite de n <= 16

gestion regroupées sous le nom de *governor*. Généralement, on distingue 5 types de governor [93] :

- le governor *Ondemand* modifie la fréquence du CPU en fonction de son utilisation. Par défaut, il ajuste la fréquence du CPU entre son niveau le plus bas Pn (lorsque l'utilisation du CPU est inférieure à un seuil prédéfini [106]) et son niveau le plus élevé P0 (lorsque la charge CPU est supérieure à une autre valeur seuil prédéfini),
- le governor *Performance* conserve toujours la fréquence du CPU à sa valeur la plus élevée peu importe la charge du CPU (état P0),
- le governor *Powersave* maintient la fréquence du CPU au plus bas niveau (état Pn),
- le governor *Conservative*, augmente ou diminue la fréquence du CPU en fonction de sa charge en la faisant passer successivement par tous les niveaux de fréquence existants,
- le governor *Userspace* permet aux utilisateurs (via des applications ou directement) de régler manuellement la fréquence du processeur.

Par défaut, le *governor Ondemand* constitue la politique de gestion exploitée.

4.2.3 Coordination entre l'ordonnanceur de VMs et le DVFS

Considérons un système hôte sur lequel fonctionnent 2 VMs (V20 et V70). Les 2 VMs sont respectivement configurées de façon à utiliser 20% et 70% du temps processeur du système hôte soit $\alpha_{v20} = 20$ et $\alpha_{v70} = 70$. Le governor ondemand, utilisé par défaut, fixe la fréquence du processeur V_t appropriée en fonction de la charge totale L_{tot} du processeur à un instant donné.
Nous supposons dans l'exemple illustratif ci-dessous que la réduction de la fréquence du processeur ralentit le processeur de moitié c'est-à-dire $V_t = V_{max}/2$ (50% de perte de vitesse d'exécution). Nous considérons ensuite les 2 scénarios suivants se différentiant essentiellement par le type d'ordonnanceur de VMs utilisé.

Scénario 1 - Ordonnanceur à capacité fixe
Nous supposons que le système hôte est configuré de façon à utiliser un ordonnanceur de VMs à capacité fixe. Supposons également que :

- la V20 est *"normalement" surchargée*. Nous entendons par *VM "normalement" surchargée* une VM qui utilise entièrement (100%) sa capacité de CPU ($\text{VMLoad}_{v20} \simeq 100 \Rightarrow L_{v20} = 20$)
- la V70 est sous-chargée c'est-à-dire qu'elle utilise 0% de sa capacité de temps ($\text{VMLoad}_{v70} \simeq 0 \Rightarrow L_{v70} = 0$)

De ce fait, notre système hôte est globalement sous-chargé et dispose d'une charge totale théorique de 20% (*qui correspond à la charge de la V20* $L_{tot} = L_{v20}$). Dans cette situation, le governor ondemand abaisse la fréquence du processeur de façon à satisfaire la charge "théorique" du système hôte. Cette réduction permet à l'évidence de réduire la consommation énergétique du processeur du système hôte.

Toutefois, la V20 n'obtient pas la totalité de ressource dont elle a besoin. En effet, d'après 4.1,

$$\begin{aligned} \beta_{v20} &= \alpha_{v20} \times \frac{V_t}{V_{max}} \\ &= 20 \times 0.5 \text{ puisque } V_t = \frac{V_{max}}{2} \end{aligned}$$

Ainsi, au lieu de recevoir sa capacité de temps de calcul, la V20 reçoit la moitié ($\beta_{v20} < \alpha_{v20}$) en raison de la réduction de la fréquence. L'économie réalisée sur la consommation énergétique du processeur a ainsi lieu au détriment de la V20 qui est grandement pénalisée.

En résumé, dans ce scénario, la baisse de fréquence du processeur impacte les performances de la V20 et empêche le respect du SLA. Ceci se justifie du fait que l'ordonnanceur de VMs n'est pas conscient du changement de fréquence du processeur effectué par le DVFS.

Scénario 2 - Ordonnanceur à capacité variable Considérons à présent que notre système hôte utilise un ordonnanceur de VMs à capacité variable. Avec les mêmes hypothèses que dans la section précédente : *la V20 "normalement" surchargée et la V70 sous-chargée*. Les tranches de temps processeur inutilisées de la V70 sont données à la V20 grâce au principe de fonctionnement de l'ordonnanceur à capacité variable. Cette redistribution compense l'effet de bord décrit avec l'ordonnanceur de VMs à capacité fixe lors de la réduction de la fréquence du CPU.
Cependant, le fait d'utiliser les tranches de temps de la V70 peut conduire à une surconsommation de la V20 par rapport à sa souscription. Cette situation empêcherait toute baisse optimale de la fréquence du processeur ($V_t > V_{max}/2$). De plus, ceci éviterait toute éventuelle opération de consolidation parce que, du point de vue du service d'équilibrage de charge, (1) la machine hôte est éventuellement surchargée et (2) la V20 trop grosse l'empêchant éventuellement d'être migrée.
Grâce donc à l'ordonnanceur à capacité variable, le SLA de la VM est respecté mais peut induire une surconsommation des ressources. Dans ce scénario, l'utilisation de l'ordonnanceur à capacité variable ne permet pas un usage optimal des ressources du fournisseur.

En raison de l'indépendance entre l'ordonnanceur de VMs et la politique des gestion du DVFS (governor), nous observons qu'il est difficile pour un gestionnaire de ressources au sein d'un IaaS de garantir à la fois le SLA et de favoriser l'économie d'énergie de l'infrastructure. En effet, soit le fournisseur ne garantit pas le SLA requis par ses clients (avec l'ordonnanceur à capacité fixe), soit les VMs seront autorisées à utiliser plus de tranches de temps processeur que celui souscrit (avec l'ordonnanceur à capacité variable) empêchant de la sorte l'ajustement du niveau fréquence du processeur.

Les scénarios précédents révèlent un réel problème de coordination entre le gestionnaire de ressources et les politiques de gestion d'énergie. Le respect du SLA et la réduction d'énergie dans un environnement virtualisé constituent donc en réalité deux exigences à la limite "conflictuelles".

4.2.4 Synthèse

Les approches de gestion d'énergie et d'ordonnancement de VMs utilisées de façon disjointe ne permettent pas à un fournisseur de service de garantir à la fois le SLA souscrit par ses clients et une utilisation efficiente de l'ensemble des ressources de son infrastructure. Bien que le DVFS permette de réduire la consommation énergétique en cas de processeur sous-chargé, son utilisation basique dans un environnement virtualisé entraîne une dégradation de la performance des VMs. Il devient donc primordial de proposer un système permettant d'exploiter les atouts qu'offrent le DVFS et la virtualisation sans pénaliser ni l'utilisateur ni le fournisseur.

Dans le chapitre suivant, nous présentons les approches de gestion de ressources en environnement virtualisé qui cherchent à s'attaquer à cette problématique.

Chapitre 5

Etat de l'art

Contents

Tout au long des chapitres précédents, nous avons observé qu'au sein d'un IaaS il existe 2 principales exigences dont la satisfaction n'est pas toujours évidente : le respect du SLA et l'économie d'énergie. Au travers d'analyses, nous avons proposé un modèle de gestion d'énergie basé (*Cf. section 3.2*) sur le principe de périphérique à vitesse variable. En section 4.1.3, nous avons observé que ces procédés de gestion d'énergie ont un impact sur la performance des périphériques/composants parce qu'ils réduisent leur vitesse d'accès/d'exécution. Cette baisse de vitesse influe par la même occasion sur les performances des applications les sollicitant. Particulièrement dans un IaaS, la performance des VMs est perturbée et les SLA demandés par les clients ne sont pas respectés.
Dans cette thèse, notre problématique est d'offrir un système de gestion de ressources, dans un environnement virtualisé, respectueux du SLA et de la consommation énergétique. Dans ce chapitre, nous définissons des critères que nous utilisons pour évaluer et comparer les projets qui se sont intéressés à cette problématique. A la suite de ces critères d'évaluation, nous présentons quelques solutions d'ordonnancement de VMs qui accordent de l'importance à la gestion d'énergie consommée par le CPU dans le but de la réduire.

5.1 Critères d'évaluation

Dans un IaaS, le fournisseur doit principalement garantir le SLA de ses clients. Plusieurs dispositifs (matériels et logiciels) sont ainsi mis en œuvre pour répondre idéalement aux souhaits des clients. Sur le plan matériel, cela se traduit par l'acquisition de plateformes de plus en plus performantes. Toutefois, on estime que la consommation énergétique des infrastructures informatiques a augmenté de 56% entre 2005 et 2010 ; et a représenté entre 1,1% et 1,5% de la consommation mondiale d'électricité en 2010 [65]. En plus des coûts d'exploitation élevés, cela se traduit par l'émission du dioxyde de carbone (CO2) estimée à 2% des émissions mondiales. Sur le plan logiciel, de nouvelles stratégies d'ordonnancement de VMs sont proposées. Dans cette section, nous ferons une description générale de quelques approches (les plus représentatives) en portant attention aux 2 critères suivants :

- **Respect de SLA** : ce critère consiste à identifier l'impact que peut avoir l'utilisation d'une politique de gestion d'énergie sur la performance des VMs. Il est question de vérifier s'ils existent des VMs pour lesquelles le SLA n'est pas respecté du fait de l'application d'une politique de gestion d'énergie,
- **Efficience énergétique** : ce critère permet d'évaluer si les ressources du fournisseur sont utilisées de façon efficiente. Il s'agit de vérifier s'il existent des VMs qui de part leur consommation en ressources pourraient empêcher la baisse du niveau de fréquence et par ricochet empêcher toute éventuelle consolidation ? En bref, s'assurer s'il y'a des VMs qui dépassent leur capacité de ressource ?

Les VMs dans un IaaS sont gérées de façon centralisée par l'ordonnanceur de VMs et de façon distribuée par le répartiteur de charge. Pour chacun des niveaux de gestion, de multiples approches de gestion d'énergie ont été proposées. Dans le cadre de cet état de l'art, nous nous sommes prioritairement intéressés aux solutions centralisées qui constituent le cœur de notre travail.

Les principales approches de gestion centralisée sont constituées entre autres des ordonnanceurs de VMs des technologies de virtualisation existantes et de quelques extensions de ces ordonnanceurs. Notre état de l'art respecte cette subdivision.

5.2 Approches basées sur des solutions existantes

5.2.1 Kernel-based Virtual Machine (KVM)

Le développement de KVM a commencé au sein de la société Qumranet par Avi Kivity et est actuellement co-maintenu par Marcelo Tosatti. KVM est intégré dans le noyau Linux sous forme de module depuis la version 2.6.201.

Principe de fonctionnement KVM [63] est une solution de virtualisation assistée par le matériel. Pour son fonctionnement, KVM exploite les instructions de virtualisation (Intel-VT [131] ou AMD-V [140]) intégrées dans les nouveaux processeurs. KVM se compose d'un module de noyau, kvm.ko, qui fournit l'infrastructure de base de la virtualisation, et d'un module spécifique au type de processeur du système de déploiement : kvm-intel.ko ou kvm-amd.ko. Les VMs déployées sur un noeud exécutant KVM sont gérées comme des processus par l'ordonnanceur par défaut de Linux nommé *Completely Fair Scheduler(CFS) [105]* auquel quelques extensions ont été ajoutées.

Sachant que les VMs sont gérées comme des processus, l'ensemble des processus qui y sont exécutés lui sont associés grâce à la notion de *groupe de contrôle (cgroup)*. La création d'un *groupe de contrôle* permet de rassembler une VM et les processus qu'elle exécute de façon à décrémenter la capacité CPU qu'utilisent les processus sur la capacité allouée à la VM. Ainsi, la capacité allouée à un VM est décomptée à chaque fois qu'un processus qui lui est associé s'exécute. Ce principe de groupe de contrôle permet ainsi de conserver le principe d'encapsulation qui caractérise les VMs.

Dans son fonctionnement, l'ordonnanceur CFS vise à repartir le processeur entre les VMs suivant une équité parfaite. Il s'agit pour CFS d'assurer un équilibre entre les durées d'exécution des VMs. Pour cela, l'ordonnanceur *CFS* se base sur un arbre binaire de *recherche rouge-noir* (Figure 5.1) et fonctionne comme suit :

- l'ensemble des VMs sont triées en fonction de leur temps d'exécution sur le processeur et sont stockées dans un arbre,
- la VM qui aura passé le plus de temps en exécution sera tout à droite de l'arbre. Ainsi, l'ordonnanceur choisit toujours la VM la plus à gauche pour la prochaine exécution,
- au terme de son quantum de temps, la VM est ensuite re-insérée dans l'arbre binaire trié. Grâce aux propriétés de l'arbre rouge-noir, le temps d'exécution entre les VMs converge vers un équilibre.

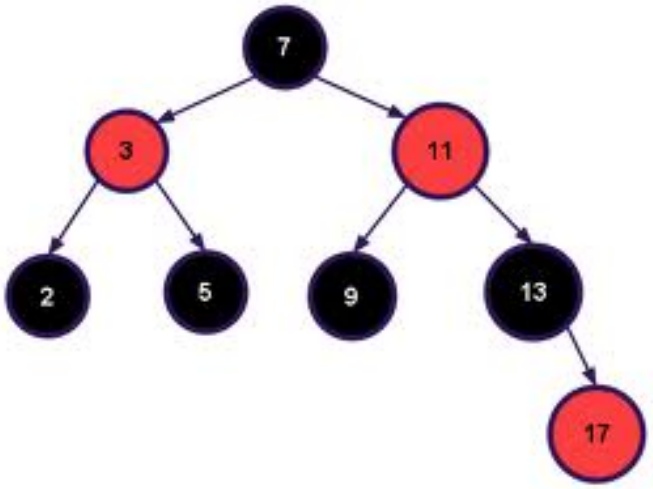

Fig 5.1. Arbre de rouge-noir de priorité des VMs

Sur la base de ce fonctionnement, si une seule VM est en cours d'exécution, elle utilisera entièrement le temps processeur (quelque soit sa capacité initiale) si elle en a besoin. L'ordonnanceur CFS fait ainsi partie de la famille des ordonnanceurs à capacité variable (*Cf. section 4.2.1* parce qu'il intègre la notion de conservation de travail (*work-conserving scheduler*).

Évaluation L'appartenance de l'ordonnanceur CFS de KVM à la famille des ordonnanceurs à capacité variable conduit à conclure qu'il ne respecte pas les critères d'évaluation que nous avons définis. En effet, dans un environnement virtualisé avec KVM, face à l'inactivité d'une ou de plusieurs VMs, leur temps CPU sera alloué aux VMs actives qui en ont besoin. Cette ré-allocation a pour résultat d'empêcher ou de limiter la baisse du niveau de fréquence du processeur. Les VMs actives auront toujours leur SLA respecté. Toutefois, si ces VMs sont "énormément" surchargées, elles utiliseront bien plus de capacité que ce qu'elles ont souscrit empêchant ainsi une utilisation optimale des ressources. Le critère d'efficience énergétique ne serait dans ce cas pas respecté.

5.2.2 Xen

Xen est une solution de virtualisation initiée par l'université de Cambridge au Royaume-Uni. Xen est actuellement distribué sous licence GNU General Public License (GPL2) et disponible sous deux formats (objet et code source).

Principe de fonctionnement Xen permet la mise en œuvre de la paravirtualisation et de la virtualisation assistée par le matériel. Xen dispose actuellement de trois ordonnanceurs nommés : *SEDF(simple earliest deadline first)*, *credit* et *credit2* et utilise par défaut l'ordonnanceur *credit*. C'est celui que nous considérons dans la suite.

Avec l'ordonnanceur *credit*, chaque processeur du système hôte gère une file d'attente de *CPU virtuel (vcpu)*[1] à exécuter. Cette file est triée par ordre décroissant de priorité (*boost*, *under*, *over*) et le prochain vcpu à exécuter est toujours pris en tête de file. Un vcpu de priorité *boost* (plus haut niveau de priorité) est un vcpu prêt à être exécuté. Il est en attente d'une opération d'E/S destinée au processus qu'il exécute. Dès le traitement de l'opération d'E/S, le vcpu est directement placé en tête de file. Un vcpu de priorité *under* possède encore du temps CPU lui permettant de s'exécuter. Un vcpu de priorité *over* a achevé sa capacité de temps CPU durant la période encours et est placé en queue de file. L'évaluation des priorités consiste à décrémenter la capacité allouée aux VMs du temps consommé par leurs vcpus. Ainsi, un vcpu peut s'exécuter pendant tout un quantum d'exécution ou être bloqué.

1. Virtual CPU

En fonction de la valeur de la capacité à une VM, l'ordonnanceur *credit* peut se comporter comme un ordonnanceur à capacité fixe ou variable comme présenté en section 4.2.1. En effet, si la capacité allouée à la VM est nulle, l'ordonnanceur fonctionne en mode capacité variable et en mode capacité fixe dans le cas contraire.

Évaluation Fonctionnant comme un ordonnanceur à capacité fixe, l'application d'une politique de gestion de DVFS avec l'ordonnanceur *credit* de Xen conduit à un non respect du SLA comme présenté en section 4.2.3. Si les VMs sont configurées de façon à utiliser l'ordonnanceur *credit* comme une ordonnanceur à capacité variable, certaines VMs pourraient utiliser beaucoup plus de ressources que ce qu'elles ont souscrit, d'où le non respect du critère d'efficience énergétique.

5.2.3 VMware ESXi

VMware ESXi [136] est l'une des solutions de virtualisation d'entreprise développée par VMware, Inc.

Principe de fonctionnement VMware ESXi permet la mise en œuvre de la paravirtualisation et de la virtualisation assistée par le matériel. Pour chacune des solutions de virtualisation adoptée, VMware ESXi n'est pas installé dans un OS, mais intègre un noyau minimal de 32Mo.

Avec VMware ESXi, l'allocation de la capacité d'une VM se traduit par la donnée d'une valeur minimale et maximale de capacité. La valeur minimale de la capacité est un pré-requis à l'exécution d'une VM. La valeur maximale ne lui est allouée qu'en cas de besoin. Particulièrement, la capacité d'une VM peut être illimitée.
Le choix de vcpu (d'une VM) à exécuter est fait en fonction de la priorité, calculée dynamiquement, de la VM. Ce calcul consiste à évaluer le ratio entre la capacité de temps consommée par les vcpus de la VM et sa capacité allouée. La VM ayant le moins consommé sa capacité est la plus prioritaire et sera exécutée au prochain quantum.
En définissant une capacité maximale (*finie*), l'ordonnanceur de VMware ESXi se comporte comme un ordonnanceur à capacité fixe. Dans le cas d'une capacité infinie, la VM consomme autant de CPU qu'elle en a besoin, ceci en fonction de la disponibilité de la ressource. Dans ce dernier cas, l'ordonnanceur de VMware ESXi se comporte comme un ordonnanceur à capacité variable.

Évaluation Il ne nous a pas été possible de trouver des documents présentant l'impact d'une politique de gestion d'énergie sur la performance des VMs lors de l'utilisation de VMware ESXi en mode ordonnanceur à capacité fixe. Des expériences

que nous avons effectuées, nous pouvons affirmer que le SLA des VMs n'est pas respecté dans ce cas de figure.

5.2.4 Microsoft Hyper-V

Hyper-V [136] est une solution de virtualisation d'entreprise développée par Microsoft.

Principe de fonctionnement Hyper-V permet la mise en œuvre de la virtualisation assistée par le matériel grâce aux extensions de virtualisation du matériel. Hyper-V met en œuvre des concepts similaires à ceux de la technologie VMware ESXi, à savoir une capacité maximale et minimale. L'ordonnanceur de VMs présent dans Hyper-V est implanté de façon similaire à l'ordonnanceur *credit* de Xen comme ordonnanceur à capacité fixe.

Évaluation De par sa conception, l'ordonnanceur VMs d'Hyper-V ne permet pas de respecter le SLA souscrit par les clients.

5.3 Approches basées sur des extensions de solution

5.3.1 Energy Management for Hypervisor-based VMs

Cette approche est le résultat d'un projet réalisé par l'université de Karlsruhe en Allemagne et financé par Intel Corporation. A cette époque, Stoess et al. [127] définissent un outil pour la gestion de l'énergie en adéquation avec l'architecture des serveurs virtualisés.

Principe de fonctionnement Les approches habituelles de gestion d'énergie présentes au cours des années *en 2007* sont basées pour la plupart sur les OS non virtualisés. Ces OS ont un contrôle total et une pleine connaissance du matériel sous-jacent. Ces atouts permettent d'ajuster directement la consommation énergétique des périphériques pour répondre aux contraintes thermiques ou énergétiques. Toutefois, la nature multi-couche des environnements virtualisés rend de telles approches inapplicables. Stoess et al. proposent un prototype d'hyperviseur intégrant une politique de gestion d'énergie.

L'ordonnanceur de VMs présent dans cet hyperviseur utilise un algorithme d'ordonnancement nommé *stride scheduling* [144, 145, 132]. Cet ordonnanceur fait partie de la famille des ordonnanceurs à capacité variable. En effet, avec l'algorithme *stride*, de type *proportionnal-share*, le partage de CPU entre les VMs est fait proportionnellement à leur poids sans limite de capacité. Ainsi, si une VM n'utilise pas son temps CPU, ce dernier est redistribué aux VMs qui en ont besoin. Cette situation empêche toute utilisation optimale de ressources notamment la consommation énergétique.
Stoess et al. proposent donc un mécanisme de gestion dynamique d'énergie basé sur l'allocation dynamique du CPU aux VMs. Ce mécanisme consiste à créer un *"CPU virtuel endormi"* rattaché à chaque CPU. Ensuite, l'hypersiveur est modifié de façon à translater directement toute demande d'arrêt virtuel de vcpu en arrêt effectif. Ainsi, si un vcpu souhaite passer en veille, son poids est réduit et son reste de temps d'exécution sera traduit par le "CPU virtuel endormi" comme une durée d'arrêt réel du processeur correspondant. Le poids du vcpu est incrémenté quand il devient actif. Toutefois, l'augmentation ou réduction du poids des vcpus est fonction de l'énergie maximale allouée à la VM. Une boucle de contrôle est invoquée périodiquement et permet de s'assurer que l'énergie consommée n'excède pas la limite énergétique dédiée à la VM.

Évaluation Cette approche peut être assimilée à un ordonnanceur à capacité fixe, dans lequel le SLA à respecter une limite de consommation énergétique. Ainsi, nous pouvons dire que cette approche respecte nos 2 critères. Cependant, elle n'est pas applicable dans le contexte du IaaS.

5.3.2 PCFS : A Power Credit based Fair Scheduler

L'approche *Power Credit based Fair Scheduler (PCFS)* est le fruit des travaux de Wen et al. [149]. Son but est de proposer un ordonnanceur de VM qui intègre la gestion de l'énergie à travers un usage efficace de diverses mesures d'économie d'énergie, en particulier le DVFS, présent dans presque tous les processeurs actuels.

Principe de fonctionnement Le but initial recherché par les ordonnanceurs de VMs est le partage équitable de ressources entre les VMs dans l'intention d'obtenir de bonnes performances. Toutefois, avec la croissance des performances des processeurs, la hausse de leur consommation énergétique et l'avènement du DVFS, l'objectif des ordonnanceurs de VMs prend actuellement en compte l'utilisation efficiente de l'énergie consommée par les VMs. Cette recherche est devenue l'un des grands défis des plateformes de virtualisation.

Wen et al. propose un ordonnanceur de VMs nommé PCFS. ce dernier est une extension de l'ordonnanceur "sched-credit" de Xen. Configuré comme un ordonnanceur à capacité fixe ou variable, il ne permet donc pas de satisfaire nos 2 critères

d'évaluation. Afin de garantir une utilisation optimale de la consommation énergétique des CPUs, PCFS propose d'intégrer les politiques de gestion du DVFS dans l'ordonnanceur.
A cet effet, l'ordonnanceur PCFS regroupe les cœurs d'un processeur en zone en fonction du niveau de fréquence (appelée *power zone*) de ceux-ci. PCFS effectue ensuite un regroupement spatial des traitements en fonction de ces zones. En effet, pendant l'exécution des VMs, PCFS monitore la charge globale des cœurs et déclenche 2 types d'actions :

- Si la charge globale d'une zone est inférieure à 20%, PCFS déplace le(s) vcpu(s) en cours d'exécution dans cette zone vers un cœur de très haut niveau de fréquence (appartenant donc à une autre zone) capable de le(s) exécuter. Le(s) cœurs ainsi libérés sont mis en état de très faible consommation ou éteints afin de réduire la consommation énergétique ; Le coeur est ensuite migré dans la "power zone" correspondante,
- Sinon, PCFS fixe un niveau de fréquence du cœur permettant de conserver sa charge globale à 90% (seuil défini). Dans cette condition, le cœur peut être migrée vers la zone de niveau de fréquence correspondant à sa nouvelle fréquence.

Évaluation Dans cette approche, l'utilisation du regroupement spatial sur les cœurs du processeur permet de réduire la consommation énergétique globale du système hôte. Cet ordonnanceur respecte donc le critère d'efficience énergétique. Toutefois, la "consolidation parfaite" n'existant pas, il est possible d'avoir des VMs qui utilisent plus de ressources que ce qu'elles ont souscrit du fait de l'utilisation d'un ordonnanceur à capacité variable.

5.3.3 VirtualPower

Nathuji et al. [97, 98] ont été les premiers à proposer une solution de gestion d'énergie pour des architectures virtualisées. Leur solution, nommée *VirtualPower*, est le fruit d'un projet de l'Institut technologique de Georgia financé conjointement par le programme de bourse NSF ITR et Intel. Le but de ce projet est double : local à un système hôte et global à un centre d'hébergement. Nous présentons le coté local de la solution.

Principe de fonctionnement Définir un ordonnanceur de VMs qui est capable d'adapter la fréquence d'exécution du CPU en fonction de la charge des VMs, tout en respectant leur SLA, est assimilable à un système hôte dans lequel la gestion d'énergie est effectuée au niveau de la VM. L'objectif recherché par l'approche *VirtualPower* est de simuler la gestion d'énergie initiée par l'OS invité.
VirtualPower dans son implémentation comme extension de Xen propose la mise en place de mécanismes logiciel et matériel permettant à l'hyperviseur de prendre en

compte [2] et d'appliquer les opérations de gestion d'énergie émises par les VMs. Ainsi, lorsqu'une VM initie une opération de gestion d'énergie, celle-ci est interceptée et transmise au module de VirtualPower chargé d'appliquer les politiques, via un canal de communication (logiciel) par lequel les opérations de gestion d'énergie des VMs sont transmises à l'hyperviseur. Si le matériel du système hôte permet l'application du DVFS, celle-ci est directement appliquée. Dans le cas contraire, l'état souhaité est émulé par l'ordonnanceur de VMs. Concrètement, au moment d'exécuter la VM concernée, l'ordonnanceur lui allouera moins (respectivement plus) de temps de processeur s'il s'agit d'une réduction (respectivement d'une augmentation) de niveau de fréquence.

Évaluation Avec cette approche, la politique de gestion d'énergie adoptée est soit une modification de fréquence, si possible, soit une modification du temps processeur de la VM dans le cas échéant.
Théoriquement, VirtualPower permet à chaque VM de fixer son niveau de fréquence sans influencer les autres tout en respectant le SLA. Toutefois, le support matériel étant partagé par plusieurs VMs, l'utilisation du DVFS peut s'avérer conflictuelle et ne pas respecter nos 2 critères voulus. Considérons les 2 VMs (V20 et V70) de notre exemple de la section 4.2.3, exécutées avec un ordonnanceur à capacité fixe. Si la V70 parce que sous-chargée demande la baisse du niveau de fréquence du CPU, le problème d'insatisfaction de la V20 demeure ; d'où le non respect du SLA.

5.4 Synthèse

Dans cette section, nous avons identifié 2 critères permettant d'évaluer les travaux relatifs à la gestion efficace de ressources dans un système virtualisé en respectant le SLA et favorisant l'économie d'énergie. Les solutions s'étant intéressées à ce problème avaient pour objectif soit d'économiser de l'énergie soit de garantir le SLA. Difficile pour la plupart d'atteindre conjointement les 2 objectifs qui semblent parfois conflictuels.

Dans le chapitre suivant, nous présentons notre approche qui permet l'atteinte de ce double objectif.

2. La VM, bien que virtuelle, est une machine "à part entière". Elle dispose des modules ACPI permettant d'agir sur l'état de performance de ses périphériques. Toutefois, à ce jour l'ensemble de ces opérations est intercepté et ignoré par l'hyperviseur.

Troisième partie

Contributions

Chapitre 6

Conception d'un Ordonnanceur de VMs de type *power-aware*

Contents

Dans le chapitre précédent, nous avons fait état de quelques travaux de recherche autour de l'ordonnancement des VMs. Nous nous sommes focalisés sur ceux ayant mis un accent sur la réduction de la consommation énergétique et le respect du SLA dans des environnements virtualisés. Bien que des progrès significatifs aient été réalisés, pour la plupart des solutions proposées, la baisse de la consommation énergétique est faite au détriment du SLA ou inversement.

Dans cette thèse, nous proposons une approche d'ordonnancement de VMs qui satisfait ces 2 objectifs. Dans ce chapitre, nous présentons l'orientation générale que nous avons adoptée lors de la conception de notre approche. De là, nous aboutissons à la conception proprement dite.

6.1 Orientation générale

Les analyses réalisées dans les chapitres précédents montrent que le but recherché par l'ensemble des politiques de gestion d'énergie est l'obtention d'un *système "éco-*

énergétique" et efficient. Nous entendons par *système "éco-énergétique"*, un système dans lequel l'énergie consommée est proportionnelle à la charge de travail. L'efficience d'un tel système se traduit par sa faculté à garantir en tout temps le SLA souscrit. Dans un système virtualisé, l'atteinte de ces buts est du ressort de l'ordonnanceur de VMs et du DVFS.
Or, en raison de l'indépendance existant entre l'ordonnanceur de VMs et les politiques de gestion du DVFS, ces 2 finalités ne sont à l'accoutumée pas atteintes. Ceci du fait que : (1) soit le fournisseur ne peut pas garantir la QoS requise par les VMs (s'il utilise un ordonnanceur à capacité fixe), (2) soit certaines VMs seront autorisées à utiliser plus de capacité CPU que celle souscrite (s'il utilise un ordonnanceur à capacité variable). Ce deuxième cas de figure empêchant la mise en œuvre des politiques du DVFS utiles pour la baisse de la consommation énergétique du système.

Idéalement, pour un usage efficient et "éco-énergétique" d'une infrastructure virtualisée, l'ordonnanceur doit être conscient de l'existence de politiques de gestion d'énergie du processeur. Cette "conscience" se traduit, par la prise en compte du niveau de fréquence courant du processeur avant toutes décisions d'ordonnancement. Plus précisément, un tel ordonnanceur doit pouvoir ré-évaluer la capacité CPU à allouer aux VMs en fonction de la valeur actuelle de fréquence du processeur. En effet, l'allocation de la capacité CPU aux VMs est basée sur la fréquence maximale. Ainsi, tout changement de fréquence doit être répercuté sur la capacité allouée à chaque VM afin de contrebalancer l'effet de la modification de la fréquence. Par conséquent, une VM verra sa capacité augmenter (respectivement diminuer) lorsque la fréquence du processeur sera abaissée (respectivement augmentée). Cette nouvelle capacité (liée à la nouvelle fréquence) doit toutefois être équivalente à la capacité initiale liée à la fréquence maximale.

Le calcul dynamique de la nouvelle capacité CPU à allouer aux VMs n'est pas trivial. Ce calcul s'appuie sur deux hypothèses principales que nous définissons comme suit :

- la ***proportionnalité entre la fréquence et la performance*** : cette propriété symbolise le fait qu'à la suite d'une modification de fréquence du processeur, l'impact sur la performance des VMs est proportionnel au changement de fréquence,
- la ***proportionnalité entre la capacité et la performance*** : cette propriété stipule qu'après toute modification de capacité CPU attribuée à une VM, l'impact sur sa performance est proportionnel au changement de capacité effectué.

Ainsi, si nous considérons le scénario présenté en section 4.1.3, lorsque la fréquence du processeur est diminuée (ralentissement du processeur de l'ordre de 50%), à la V20 il sera donné 40% de capacité pour compenser la réduction de la fréquence.

L'approche que nous proposons dans le cadre de cette thèse vise, d'une part à définir les principaux éléments utiles à la mise en place d'ordonnanceurs de VMs

conscients du niveau de fréquence d'exécution du processeur. D'autre part, elle vise à la mise en place d'ordonnanceurs de VMs qui, en plus d'assurer une gestion équitable du processeur, veille à réduire au mieux sa consommation énergétique. Cette famille d'ordonnanceur est souvent reprise dans la littérature sous le terme de : *Energy/Power-aware scheduler (PAS)*.

6.2 Conception

Dans cette section, nous présentons les principes de notre ordonnanceur de VMs de type PAS que nous nommons *pas-sla-scheduler*. Nous commençons par présenter le but de notre approche ainsi que les conditions de sa mise en œuvre. Nous présentons ensuite le fonctionnement général de notre approche.

6.2.1 Spécification

Comme tout ordonnanceur, les premiers objectifs visés par *pas-sla-scheduler* se regroupent en la maximisation de l'utilisation du processeur et la minimisation du temps de réponse des processus. En complément, *pas-sla-scheduler* a été conçu pour répondre aux incompatibilités relevées dans le fonctionnement indépendant du DVFS et de l'ordonnanceur dans le contexte d'un système virtualisé. *pas-sla-scheduler* consiste à allouer dynamiquement plus ou moins de capacité CPU aux VMs en fonction de la fréquence courante du système.
Notre ordonnanceur *pas-sla-scheduler* a également été développé pour permettre une exploitation efficiente de l'infrastructure en terme de consommation énergétique. Il s'agit ici d'attribuer au processeur un niveau de fréquence lui permettant d'absorber la charge des VMs sans dégrader leur performance, tout en minimisant le gaspillage énergétique.

Afin que notre approche soit exploitable au sein d'un IaaS, *pas-sla-scheduler* doit respecter les contraintes d'implémentation suivantes : être *non intrusivif*, *transparent aux applications* et *transparent à l'architecture* :

1. La *non intrusivité* de *pas-sla-scheduler* se traduit par l'absence d'éventuelles modifications des OS invités. De plus, *pas-sla-scheduler* doit être facilement intégrable et exploitable au sein des OS standards. La performance des applications ne doit pas être dégradée du fait de l'utilisation de *pas-sla-scheduler*.
2. *pas-sla-scheduler* se veut transparent aux applications exécutées dans les VMs. Quelque soit l'application exécutée dans un environnement virtualisé dans lequel *pas-sla-scheduler* est déployé, aucun changement [1] (aussi léger soit il) ne doit lui être apporté.

1. Changement du type compilation, sachant que l'aspect configuration n'est pas considéré comme un changement

3. *pas-sla-scheduler* se doit également d'être transparent à l'architecture matérielle. Quelque soit le type d'architecture adoptée, le principe de fonctionnement de *pas-sla-scheduler* est tenu d'être identique afin d'en faire abstraction.

Afin de respecter ces critères, plusieurs niveaux d'implémentation ont fait l'objet de notre étude :

- *Niveau utilisateur pour la gestion de capacité.* Dans cette conception, la gestion du niveau de fréquence du processeur est entièrement assurée par le système de gestion du DVFS standard à travers sa politique par défaut *ondemand*. A cette gestion, est intégrée une application exécutée dans le système hôte chargée (i) de monitorer la fréquence courante du processeur, (ii) de calculer la nouvelle capacité de temps CPU de chaque VM (correspondante à la fréquence courante) de façon à garantir la capacité initialement allouée et (iii) de la leur attribuer.
- *Niveau utilisateur pour la gestion de la fréquence et de la capacité.* Dans cette conception, le DVFS est géré par une application via le governor "userspace". Cette application s'exécute dans le système hôte et monitore les charges des VMs. Périodiquement, l'application calcule et détermine la fréquence du processeur capable d'absorber la charge du CPU sans dégradation de la QoS. Ensuite, elle calcule et alloue les nouvelles capacités de temps CPU aux VMs.
- *Niveau système pour la gestion de la fréquence et de la capacité.* Dans cette dernière conception, l'approche est implémentée à l'intérieur de l'hyperviseur, comme un ordonnanceur de VMs à part entière. La détermination de la nouvelle fréquence du processeur et le calcul des capacités des VMs sont effectuées par l'ordonnanceur à chaque décision d'ordonnancement.

Nous avons effectué diverses expérimentations sur la base de ces trois niveaux d'implémentation. La quasi totalité des tests effectués à l'aide des implémentations au niveau utilisateur s'est avérée très intrusive du fait de l'usage des appels systèmes et du manque de réactivité de l'application de calcul. Nous avons donc choisi d'implanter notre approche au niveau système. Bien que certains tests aient été effectués pour chacune des options d'implantation, les résultats présentés dans ce document sont basés sur le troisième choix de mise en œuvre.

6.2.2 Fonctionnement général

L'architecture globale de *pas-sla-scheduler* telle que présentée par la figure 6.1, montre un ordonnanceur de VMs ayant la possibilité d'agir sur la fréquence du processeur et sur la capacité CPU des VMs. Cette vue globale permet de distinguer 2 catégories d'entités avec lesquelles *pas-sla-scheduler* interagit : le processeur et la VM. Cette représentation architecturale met également en évidence les 2 principaux services de *pas-sla-scheduler* qui permettent d'agir sur les processeurs et les VMs.
Avant de détailler les spécificités de ces composants, nous allons définir quelques concepts que nous utilisons dans la suite.

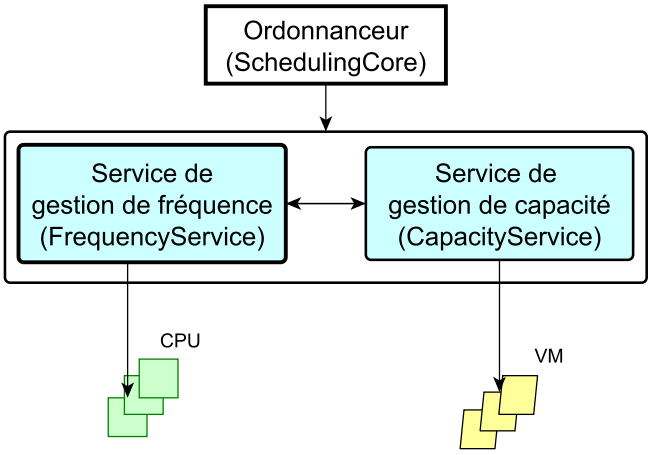

Fig 6.1. Architecture globale de *pas-sla-scheduler*

6.2.2.1 Définition des concepts

Dans la suite de ce document, nous utilisons en fonction des cas, les concepts suivants :

- **La *charge relative*** est la charge obtenue lors des opérations de monitoring du processeur. Elle est implicitement liée à la fréquence courante du processeur
- **La *charge absolue*** est la charge du processeur qu'on aurait eu si le processeur avait été à sa fréquence maximale
- **VM "normalement" chargée** est une VM dont la charge CPU est supérieure ou égale à un seuil de charge prédéfini
- **VM en "trashing"** est une VM qui génère une charge supérieure à 100%. Elle n'est certes pas visible lors du monitoring de la VM, mais elle se traduit par une baisse de performance de la VM parce que cette dernière a besoin de ressource non existante

6.2.2.2 Fonctionnement des entités

Les entités avec lesquelles *pas-sla-scheduler* interagit durant son fonctionnement sont : *le processeur et la VM.*

Le processeur Alternativement, le processeur exécute des opérations initiées par les VMs, celles utiles pour son propre fonctionnement ou les instructions de l'hyperviseur. Au sein d'un système virtualisé, le processeur se caractérise par une charge, un niveau de fréquence, une file d'attente de CPU virtuel (*vcpu* [2]) à exécuter et un ensemble d'algorithmes d'ordonnancement.

2. Virtual CPU

Dans les processeurs actuels [3], la charge du processeur et le niveau de fréquence sont étroitement liés. Typiquement, la charge du processeur est fonction des demandes des VMs mais varie aussi suivant son niveau de fréquence. L'allocation de la fréquence du processeur est faite conformément à la politique governor configurée au sein du DVFS. Le governor *ondemand* [4] permet de définir pour chaque processeur une charge maximale *up_threshold*. Si ce seuil est atteint, la fréquence du processeur est immédiatement fixée à sa valeur maximale. Dans le cas contraire, elle est fixée à sa fréquence minimale. La charge du processeur variant en fonction du niveau de fréquence, si cette dernière est augmentée, la charge du processeur est inversement impactée ; et vice-versa. Ces changements influent sur la consommation énergétique du processeur.

Fort de l'influence que peut avoir la fréquence du processeur sur sa charge, le *pas-sla-scheduler* intègre un service de gestion de fréquence qui agit de façon "intelligente" sur le processeur. L'ordonnanceur *pas-sla-scheduler* se sert des rapports de proportionnalité définis en section 6.1 pour identifier le niveau de fréquence satisfaisant à la charge du CPU. Cette satisfaction consiste à identifier le niveau de fréquence seuil à allouer à un processeur sans engendrer de perte de performance ou de gaspillage de ressources.

La VM La VM est une encapsulation logicielle semblable à un système standard (OS et applications). De ce fait, elle dispose de tous les périphériques nécessaires pour son fonctionnement, notamment le processeur appelé *vcpu*. A l'image du fonctionnement du processeur décrit plus haut, le vcpu dispose d'une charge, d'une file d'attente de processus à exécuter et des algorithmes d'ordonnancement. Soulignons qu'à ce jour, les éléments relatifs au niveau de fréquence du processeur et du DVFS ne sont pas encore présents au sein des VMs. Toutefois, certains travaux de recherches se sont intéressés à simuler le comportement du DVFS au sein des VMs [**?**].
Au sein de la VM, l'ordonnanceur (en fonction de l'algorithme mis en œuvre) choisit un processus et l'exécute sur un vcpu de sorte que l'utilisation de l'ensemble des vcpu de la VM soit optimale. La charge des vcpu d'une VM ne représente qu'une portion de la charge du processeur réel sous-jacent, du fait de la capacité CPU allouée à chaque VM. Or la capacité de la VM étant allouée sur la base de la fréquence maximale du processeur du système hôte, les opérations de baisse de fréquence entrainent une montée de la charge CPU. Cette situation peut avoir un impact assez important sur le SLA d'une VM si celle-ci se passe en mode "trashing".

Afin de ne pas pénaliser les VMs ainsi concernées, *pas-sla-scheduler* intègre un service de contrôle de capacité qui ajuste la capacité allouée à une VM lors des baisses/hausses de fréquence.

3. Processeur intégrant la technologie DVFS
4. Configuré par défaut

6.2.2.3 Fonctionnement des services

pas-sla-scheduler est un ordonnanceur de VMs de la famille des ordonnanceurs à capacité fixe. Au cours de son fonctionnement, *pas-sla-scheduler* alloue de façon équitable le processeur entre les vcpus proportionnellement aux capacités de chaque VM. Il est constitué de 2 services qui contribuent à résoudre le problème de coordination constaté entre le DVFS et l'ordonnanceur de VMs dans les systèmes virtualisés (présenté en section 4.2.3).
Nous présentons dans cette section le fonctionnement général de ces 2 services, essentiels pour la mise en place de *pas-sla-scheduler*. Nous définissons au préalable quelques concepts clés.

Définition Pour décrire chacun des services de notre ordonnanceur *pas-sla-scheduler*, nous utilisons le pattern *Sonde - Module décisionnel - Effecteur* (que nous abrégeons par *SME* par la suite) comme élément de structuration (Figure 6.2) : :

- *Sonde* : Elle permet d'avoir une vue sur la ressource monitorée. Elle scrute continuellement l'état de la ressource et retourne ses valeurs.
- *Module décisionnel* : sur la base des valeurs retournées par la sonde, ce module vérifie l'ensemble des pré-conditions et effectue toutes les opérations fonctionnelles liées au service concerné.
- *Effecteur* : Il permet au service de réaliser effectivement l'action qui le définit. Il intervient au terme du module décisionnel et a un accès exclusif à la ressource afin de la laisser dans un état cohérent.

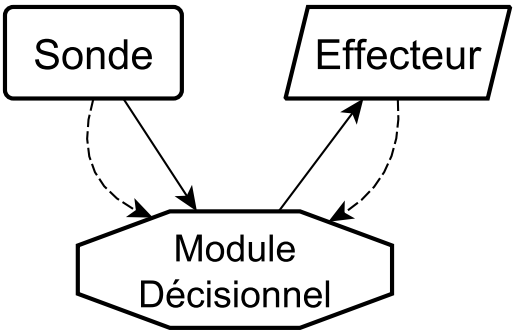

Fig 6.2. Principe de fonctionnement SME

Service de gestion de fréquence Le *FrequencyService*, intégré dans *pas-sla-scheduler*, est en charge de la gestion du niveau de fréquence du processeur. Il est constitué de 3 principaux modules *CPUMonitor*, *FrequencyEvaluation*, *FrequencyAllocation* opérant respectivement suivant le modèle SME.
Initialement, le *CPUMonitor* monitore et sauvegarde périodiquement la charge et la fréquence courante du processeur. La moyenne arithmétique des 3 dernières charges sauvegardées est utilisée comme charge relative courante du processeur par le module *FrequencyEvaluation*. Ce dernier se sert de cette charge relative, de la fréquence

courante et maximale du processeur pour évaluer sa charge absolue. Cette évaluation consiste à déterminer la charge équivalente à la charge relative à fréquence maximale. Ensuite, le module *FrequencyEvaluation* détermine le niveau de fréquence approprié pour la charge absolue obtenue. Le mécanisme de détermination du niveau adéquat de fréquence du processeur consiste à sélectionner, parmi les niveaux de fréquence présents sur le processeur, la plus petite fréquence capable d'absorber la charge absolue du processeur sans perte (ou une légère perte) de performance. Au terme de cette opération, la nouvelle fréquence choisie est allouée au processeur concerné grâce au module *FrequencyAllocation*.
Au terme de ces opérations, la fréquence courante est adéquate à la charge du CPU et la mise à jour des capacités de VMs est prête pour sa mise en œuvre. Ce service fait l'objet du paragraphe suivant.

Service de gestion de capacité Le *CapacityService* de notre ordonnanceur *pas-sla-scheduler* a pour objectif de résoudre le problème de détérioration de SLA observé au sein des ordonnanceurs de VMs à capacité fixe. Il utilise la fréquence courante du processeur fournie par le *FrequencyService* pour calculer les nouvelles valeurs de capacité des VMs. De façon analogue, le *CapacityService* est composé de 3 modules *FrequencyChecker*, *NewVMCapacityComputing* et *NewVMCapacityAllocation* conforme au modèle SME.
Le *FrequencyChecker* scrute l'état du CPU présent sur le système et récupère sa fréquence courante. Le *NewVMCapacityComputing* se base sur la fréquence courante du CPU précédemment obtenue pour recalculer les nouvelles valeurs de capacité à allouer à chaque VM. Cette nouvelle capacité à allouer à chaque VM doit correspondre au maximum à la capacité qui lui a été initialement allouée lors de son démarrage (cette valeur est automatiquement sauvegardée par *pas-sla-scheduler*). Le mécanisme de calcul de cette nouvelle capacité consiste à déterminer la capacité à ajouter à la capacité initiale de la VM afin d'annuler l'effet de bord dû au changement de fréquence du CPU. Au terme de cette ré-évaluation, le module *NewVMCapacityAllocation* modifie les capacités CPU des VMs.

D'après notre étude, la résolution du problème d'incompatibilité précédemment identifié passe par la mise en œuvre d'ordonnanceur de VMs conscient de la fréquence d'exécution du processeur d'une part. D'autre part, elle nécessite de pouvoir ajuster dynamiquement la capacité de CPU allouée aux VMs. *pas-sla-scheduler* de par sa conception respecte ces 2 recommandations. Grâce à son service *FrequencyService*, l'ordonnanceur est capable d'identifier la fréquence courante du processeur et de l'adapter à sa charge. A l'aide du service *CapacitycyService*, l'ordonnanceur est informé des modifications de fréquence et réadapte les capacités de CPU allouées aux VMs. Ces opérations effectuées à la volée permettent de garantir en temps réel une bonne QoS des VMs et d'optimiser la consommation énergétique du processeur.

6.3 Synthèse

Dans ce chapitre, nous avons présenté l'approche générale de fonctionnement d'un ordonnanceur de VMs de type power-aware. Nous avons ensuite présenté les spécifications fonctionnelles, les conditions d'implantation et les principaux services caractéristiques de notre approche d'ordonnancement *pas-sla-scheduler*. Ce chapitre s'achève en relevant comment la conception que nous avons faite propose une ébauche de solution face au problème de coordination entre les politiques de gestion d'énergie et le gestionnaire de ressources.

Dans le chapitre suivant, nous détaillons l'implémentation des différents services qu'offre *pas-sla-scheduler*.

Chapitre 7

Implémentation de l'ordonnaceur de VMs *pas-sla-scheduler*

Contents

Dans ce chapitre, nous présentons de façon détaillée l'implémentation de notre approche. Premièrement, nous détaillons les différents rapports de proportionnalités utilisés lors du calcul de la fréquence du processeur et de la capacité des VMs. En second lieu, nous présentons l'environnement et les conditions de mise en œuvre de notre ordonnanceur *pas-sla-scheduler* en mettant en évidence la caractérisation des VMs et le fonctionnement de la partie ordonnancement. Ensuite, nous présentons les 2 services de *pas-sla-scheduler* qui permettent d'atteindre les objectifs souhaités par notre approche. Enfin, nous présentons comment *pas-sla-scheduler* est utilisé en environnement réel.

7.1 Rapport de proportionnalité

En comparaison aux autres ordonnanceurs de VMs, l'apport de *pas-sla-scheduler* est l'intégration des mécanismes lui permettant de modifier dynamiquement la capacité des VMs et la fréquence du processeur. Ceci est possible grâce à la définition de 2 rapports de proportionnalité (1) entre la performance d'une VM (temps d'exécution ou charge CPU) et la fréquence du processeur réel sous-jacent ; et (2) entre la capacité de la VM et la fréquence du processeur sur lequel elle est exécutée. Notons que ces hypothèses de proportionnalité sont vérifiées dans la suite du document.

7.1.1 Proportionnalité : Performance et Fréquence

Cette propriété stipule que pour toute modification de fréquence du processeur, l'impact sur les performances des VMs doit être proportionnel au changement de fréquence effectué.
La performance d'une application, selon son type, est très souvent évaluée par son temps d'exécution ou le débit de traitement des requêtes (lié à la charge du CPU). Dans un système virtualisé, cette performance dépend à la fois de la capacité de CPU dédiée à la VM et de la fréquence du processeur. Sur cette base, nous définissons les 2 rapports de proportionnalités suivant :

1. Nous définissons un premier rapport entre **le ratio de fréquence du processeur et le ratio de charge CPU** par l'équation 7.1

$$\frac{L_{max}}{L_i} = \frac{F_i}{F_{max}} \times cfl_i \quad (\text{cfl}_i \simeq 1) \tag{7.1}$$

   dans laquelle F_{max} et F_i représentent respectivement la fréquence maximale et une fréquence quelconque du processeur ; L_{max} et L_i la charge du processeur à F_{max} et F_i respectivement et cfl_i le coefficient de proportionnalité qui lie le ratio de fréquence et le ratio de charge CPU.
   Ce rapport nous montre que si nous faisons passer la fréquence du processeur de sa valeur maximale F_{max} à une valeur de fréquence inférieure F_i, sa charge CPU augmente proportionnellement de L_{max} à L_i. Les machines n'étant en général pas "éco-énergétique", nous avons introduit un coefficient de proportionnalité cfl_i entre les ratios de fréquence et de charge du processeur. Le calcul de cfl_i obtenu par expériences successives est détaillé en section 8.3.

   Pour illustration, supposons par exemple un système virtualisé ayant une fréquence maximale $F_{max} = 3000$ MHz sur lequel est exécuté une VM. Si la fréquence est abaissée à une valeur inférieure $F_i = 1500$ MHz, nous obtenons un ratio de fréquence $\frac{F_i}{F_{max}} = 0,5$. Ce ratio indique que le processeur à la fréquence F_i tourne de 50% plus lentement comparé à F_{max}. Ainsi, si nous considérons une charge courante L_{max} de 10% à fréquence maximale F_{max}, la

nouvelle charge L_i avoisinera $L_i = \frac{10}{0.5 \times cfl_i}$ ($L_i = 20\%$ *si* $\text{cfl}_i = 1$), au moins 2 fois supérieure à la charge initiale.

2. Nous établissons un second rapport entre **le ratio de fréquence du processeur et le ratio de durée d'exécution** d'une application *Appli* de type CPU-intensive dans une VM par l'équation 7.2 :

$$\frac{T^j_{max}}{T^j_i} = \frac{F_i}{F_{max}} \times cf_i \tag{7.2}$$

De même que dans l'équation précédente (Equation 7.1), F_{max} et F_i représentent respectivement la fréquence maximale et une fréquence quelconque du processeur. T^j_{max} et T^j_i représentent la durée d'exécution de l'application *Appli* dans une VM de capacité C^j respectivement à fréquence F_{max} et F_i du processeur. cf_i représente le coefficient de proportionnalité qui lie la durée d'exécution à la fréquence du processeur.

Cette équation nous apprend que la durée d'exécution d'une application (exécutée dans une VM) dépend de la capacité (représentée ici par l'indice j) allouée à la VM et de la fréquence du processeur. De même que dans le cas de la charge CPU, si la fréquence passe de sa valeur maximale F_{max} à une valeur de fréquence inférieure F_i, la durée d'exécution de l'application augmente proportionnellement de T^j_{max} à T^j_i.

Dans la suite de ce document, nous utilisons l'expression $ratio_i$ pour représenter le ratio de fréquence ($ratio_i = F_i/F_{max}$) utilisée dans l'ensemble des rapports de proportionnalités que nous décrivons. Notons également que dans la suite, la fréquence du processeur est représentée en indice et la capacité d'une VM en exposant.

7.1.2 Proportionnalité : Capacité et Performance

La performance d'une application dépend entre autre de la capacité allouée à la VM qui l'exécute. La définition de notre troisième propriété (définie par l'équation 7.3) permet d'évaluer l'influence de la modification de la capacité CPU de la VM sur les performances des applications qu'elle exécute. Cette propriété stipule que pour toute modification de la capacité attribuée à une VM, l'impact sur la durée d'exécution est proportionnelle au changement de capacité . Elle se traduit par l'équation 7.3 suivante :

$$\frac{T^{init}_i}{T^j_i} = \frac{C^j}{C^{init}} \times cft_i \quad (\text{cft}_i \simeq 1) \tag{7.3}$$

dans laquelle C^{init} et C^j représente respectivement la capacité initiale et une capacité quelconque d'une VM. T^{init}_i et T^j_i représentent la durée d'exécution d'une application *Appli* dans une VM de capacité C^{init} et C^j respectivement, les 2 à fréquence F_i du

processeur.
Ce rapport nous montre que si nous modifions la capacité d'une VM de C^{init} à C^{j}, sa durée d'exécution changera proportionnellement de T_i^{init} à T_i^{j}. Soulignons que ce rapport peut dépendre de la fréquence du CPU (identifié par l'indice i) d'où l'ajout du coefficient cft_i.

Par exemple, si la capacité allouée à une VM est augmentée de 10% à 20%, son temps d'utilisation du processeur est au minimum doublée. De ce fait, la durée d'exécution de l'application qu'elle exécute sera approximativement réduite de moitié (par rapport à la durée d'exécution lors d'une capacité de 10%) si elle est exécutée au même niveau de fréquence.

La validation de ces rapports de proportionnalité est détaillée dans le paragraphe 8.3.

7.1.3 Mecanismes de calcul de coefficient

Le calcul des coefficients cfl_i, cft_i, cf_i est effectué par expériences successives. En effet, pour chaque benchmark choisi (*Cf. section 8.1*), nous l'exécutons à plusieurs reprises pour chaque niveau de fréquence du processeur et, utilisons la valeur moyenne des résultats pour nos calculs de coefficients. En fonction du coefficient à déterminer, nous avons procédé comme suit :

- *Calcul de* cfl_i : nous avons fait varier le profil de charge à injecter sur notre benchmark et avons enregistré la charge CPU générée par chaque injection. Notons qu'ici la VM n'avait aucune limitation de capacité,
- *Calcul de* cft_i : nous avons alloué une capacité initiale à la VM et avons évalué la durée d'exécution de notre benchmark. Nous avons ensuite exécuté notre benchmark avec d'autres valeurs de capacité et avons sauvegardé les durées d'exécution obtenues,
- *Calcul de* cf_i : A l'opposé de l'expérience précédente, ici nous avons fixé la valeur de capacité initiale de la VM et avons fait varier le niveau de fréquence pour évaluer la durée d'exécution de notre benchmark.

7.2 Opération d'ordonnancement

L'architecture interne de *pas-sla-scheduler* est présentée par la figure 7.1. Elle montre le fonctionnement logique de *pas-sla-scheduler* en distinguant les services de contrôle de SLA (cadre bleu) des services d'ordonnancement proprement dit (cadre rouge). *pas-sla-scheduler* a été implanté dans Xen (*dans sa version 4.1.2*). Ainsi, afin de tirer profit des acquis de performance et de répartition de charge entre les processeurs de l'ordonnanceur par défaut de Xen (*credit scheduler*), le module ordonnancement (*SchedulingCore*) de *pas-sla-scheduler* lui est en grande majorité sem-

blable. Dans cette partie, nous présentons *SchedulingCore* en nous attardant sur les composants du *credit scheduler* de Xen qui ont fait l'objet d'une extension.

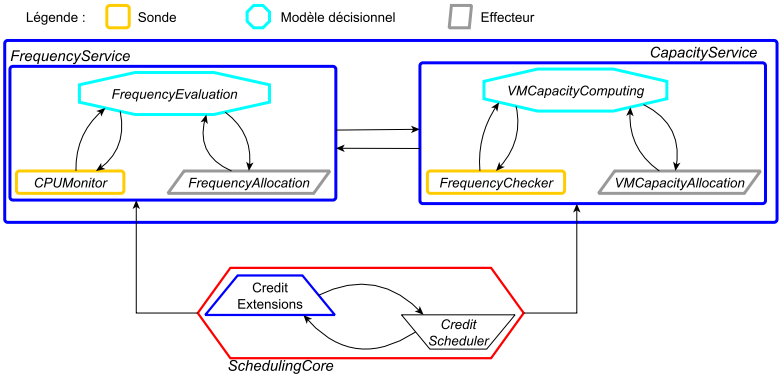

Fig 7.1. Architecture détaillée de *pas-sla-scheduler*

La phase d'initialisation de *SchedulingCore* concerne aussi bien le système hôte que les VMs. La phase d'initialisation du système hôte permet : (i) de sauvegarder les informations caractéristiques du matériel telles que : le nombre de CPU, la fréquence maximale ; (ii) d'initialiser le module de gestion de fréquence du processeur ; (iii) d'initialiser toutes la structure relative à l'ensemble des niveaux de fréquence existants (*Cf. Structure de données N° 1*) ainsi que des coefficients de proportionnalités de chacun d'eux (détaillé au paragraphe 7.3.2). La phase d'initialisation des VMs est une initialisation de la structure descriptive d'une VM étendue pour *pas-sla-scheduler* (*Cf. Structure de données N° 2*) dans le but de garder une trace de la capacité de CPU souscrite par la VM.

Structure de données 1. Processeur physique

```
struct csched_pcpu {
    struct list_head runq;
    uint32_t runq_sort_last;
    struct timer ticker;
    unsigned int tick;
    unsigned int idle_bias;
    uint64_t prev_time;
    uint64_t prev_idle;
    uint64_t avail_freqs[
       FREQ_LEVEL];
    uint16_t numfreqs;
    uint32_t load_queue[LOAD_AVG
       ];
    uint32_t last_queue_pos;
  struct {uint64_t current_freq;
        uint16_t proport_coef;
    } csched_pcpu_coef;
}
```

Structure de données 2. Machine virtuelle

```
    struct csched_dom {
    struct list_head active_vcpu
       ;
    struct list_head
       active_sdom_elem;
    struct domain *dom;
    uint16_t active_vcpu_count;
    uint16_t weight;
    uint16_t cap;
    uint16_t subscribed_cap;
    uint64_t prev_cpu_time;
    uint64_t avg_vm_load[
       AVERAGE_VALUE];
    uint16_t real_cpu_load;
}
```

Une fois le système initialisé et les VMs démarrées, celles-ci s'exécutent sur le processeur disponible en fonction de leur demande et dans la limite de la capacité qui leur est allouée. L'ensemble des opérations de choix de vcpus, de migration de vcpus entre les processeurs, de calcul de la capacité consommée et des restrictions dans l'utilisation du processeur est identique à *credit scheduler* de Xen [31].

7.3 Service de gestion de fréquence

Les différents modules du *FrequencyService* interagissent pour déterminer et fixer le niveau de fréquence adéquat à la charge du CPU. Ce service est sollicité à chaque top horloge.

7.3.1 Le module *CPUMonitor*

Périodiquement, ce module évalue la charge relative du processeur physique comme présenté par l'algorithme 3. Cette charge est obtenue en évaluant la différence entre le temps de travail ($work\_time$) et le temps d'inactivité ($idle\_time$) du processeur entre 2 points de contrôle successifs. Cette valeur n'étant pas assez représentative de l'état du système, la charge retournée par *CPUMonitor* est une moyenne numérique (ligne 17) des 3 précédentes charges du processeur sauvegardées [1] .

1. Valeur choisie après simulations

Algorithme 3. Calcul de la charge relative d'un CPU physique

```
1  static uint64_t csched_dynfreq_compute_work_load(struct csched_pcpu
       *spc, unsigned int tick, int cpuid){
2    uint64_t total_load,idle_time,current_time,work_time,
       difference_time,difference_idle,avg_load;
3    current_time = NOW();
4    //get time difference between now and last checkpoint
5      difference_time = current_time - spc->previous_time;
6      spc->previous_time = current_time;
7    idle_time = get_cpu_idle_time(cpuid);
8      //get time difference between current idle time and last
         checkpoint
9    difference_idle = idle_time - previous_idle;
10     spc->previous_idle = idle_time;
11   if (difference_time > 0){// calculate work time and then PCPU
       load
12         work_time = difference_time - difference_idle;
13       total_load = 100 * work_time/difference_time;
14   }
15     // save it to the list
16     csched_dynfreq_save_load(spc, total_load);
17     avg_load = csched_dynfreq_avg_load(spc);
18     return avg_load;
19 }
```

Dans la foulée, ce module récupère également la fréquence courante du processeur qu'il transmet au module *FrequencyEvaluation* en même temps que sa charge relative.

7.3.2 Le module *FrequencyEvaluation*

Dans un idéal absolu, la répercussion d'un changement de fréquence dans un système devrait être inversement proportionnelle à la charge du processeur (c'est à dire $\text{cfl}_i = 1$) en terme de ratio. De même, l'impact de la variation de la capacité d'une VM devrait être inversement proportionnelle à la durée d'exécution d'une application présente dans la VM (c'est à dire $\text{cft}_i = 1$). A regret, nous constatons que ce n'est pas toujours le cas. Les expériences que nous avons effectuées pour les mêmes scénarios, dans les mêmes conditions sur des architectures différentes l'ont révélées et les résultats sont présentés en section 8.2.2.

Méthode de calcul A l'aide de la charge relative et de la fréquence courante du processeur physique, le module *FrequencyEvaluation* détermine la charge absolue du processeur ainsi que sa nouvelle fréquence "éco-énergétique". Nous rappelons que la *charge absolue* L_{abs} du processeur est la charge obtenue à fréquence F_{max} équivalente à une charge relative donnée L_i à fréquence F_i. Nos mécanismes de calcul de charge absolue et de fréquence ont été regroupés sous 2 méthodes : ***méthode par***

processeur et ***méthode par processeur et fréquence*** que nous détaillons ci dessous. Ces méthodes sont basées sur les rapports de proportionnalité définis au paragraphe 7.1.

- **Méthode par processeur** : elle consiste à admettre que : (1) les coefficients cfl_i et cft_i sont identiques et égaux à 1 et ; (2) pour chaque type de processeur, le coefficient cf_i est unique pour toutes les fréquence F_i.
- **Méthode par processeur et par fréquence** : Une analyse comparative entre les résultats obtenus grâce à la "méthode par processeur" et l'utilisation du governor performance, a montré que les coefficients varient en fonction de la fréquence F_i. La *méthode par processeur et par fréquence* consiste à discrétiser le coefficient cf_i pour chaque valeur de fréquence du processeur. L'unicité des coefficients de proportionnalité utilisé jusqu'à présent a été remplacé par l'adoption d'un coefficient rattaché à chaque niveau de fréquence.

Dans la partie dédiée à l'évaluation, les résultats présentés sont essentiellement basés sur la *méthode par processeur et par fréquence* pour le calcul du coefficient de proportionnalité.

Calcul de la fréquence du processeur L'algorithme d'évaluation de la nouvelle fréquence à allouer au processeur est basé sur la charge absolue de celui-ci. Pour cela, l'équation 7.1 est utilisée. A la suite du monitoring, la charge relative (L_i) et la fréquence courante du processeur (F_i) sont utilisées pour le calcul de la charge absolue (L_{abs}) du processeur. La charge relative utilisée dans ce calcul est majorée de 10% ($L_i^{'} = 1.1 * L_i$) afin d'éviter d'avoir des VMs qui passent en mode "trashing" à la suite d'une variaton de fréquence. Ainsi, la charge absolue s'obtient grâce à l'équation suivante :

$$L_{abs} = L_i^{'} * ratio_i * \text{cfl}_i \tag{7.4}$$

Déterminer la nouvelle fréquence du processeur consiste à vérifier si la charge absolue ainsi obtenue peut être absorbée par une fréquence différente. Cette vérification s'effectue par comparaisons successives comme suit :

- Nous voulons vérifier si un CPU à fréquence F_j peut absorber la charge L_{abs}
- Nous cherchons la *charge absolue fictive* (L_{absFic}) correspondant à 100% du CPU à fréquence F_j : $L_{absFic} = 100 * ratio_j * \text{cfl}_j$
- Nous comparons cette charge absolue fictive (L_{absFic}) à la charge absolue réelle (L_{abs})
- Si $L_{absFic} < L_{abs}$, une nouvelle valeur de charge absolue fictive et une comparaison à la charge absolue sont effectuées avec les autres valeurs (inférieures) de fréquence, sinon la fréquence F_j est choisie.
- Nous supposons un processeur surchargé (charge CPU = 100%) s'exécutant à la fréquence F_j,
- Pour chaque fréquence F_j du processeur, nous comparons la *charge fictive absolue* (L_{absFic})de ce processeur à fréquence F_j avec la charge absolue réelle.

La *charge fictive absolue* est obtenue à l'aide de l'équation 7.4 dans laquelle $L_j^{'}$ = 100. Ainsi, $L_{absFic} = 100 * ratio_j * \text{cfl}_j$,

– Si la charge absolue fictive est inférieure à la charge absolue réelle, la comparaison est effectuée avec les autres valeurs de fréquence, sinon la fréquence F_j est choisie.

L'idée sous-jacente à ce calcul est de supposer un système surchargé et déterminer le niveau de fréquence pouvant l'absorber.

7.3.3 Le module *FrequencyAllocation*

La mise en œuvre de ce module nécessite de configurer le DVFS de façon à permettre la modification de la fréquence du processeur par notre ordonnanceur. Au terme du calcul de la nouvelle fréquence du processeur (section 7.3.2) et des nouvelles capacités de VM (section 7.4.2), le module *FrequencyAllocation* met à jour la fréquence du processeur si nécessaire. Cette opération est effectuée au terme des opérations d'allocation de nouvelle capacité aux VMs.

7.4 Service de gestion de capacité

De même que le service précédent, la gestion de la capacité des VMs s'effectue à chaque top horloge de l'ordonnanceur de VMs.

7.4.1 Le module *FrequencyChecker*

L'objectif général de *pas-sla-scheduler* est d'éviter tout gaspillage (de quelque nature que ce soit). Les nouvelles capacités à allouer aux VMs doivent correspondre aux capacités initialement souscrites. Ce module permet à chaque top horloge de collecter la fréquence courante du processeur.

7.4.2 Le module *NewVMCapacityComputing*

Le calcul des nouvelles capacités des VMs est basé sur les 2 autres équations énoncées plus haut (7.2 et 7.3). Cette modification des capacités des VMs vise une compensation de la perte de performance encourue par une réduction de la fréquence du processeur.

Nous détaillons le fonctionnement de ce module en nous basant sur un cas d'utilisation. Supposons une VM à laquelle est allouée une capacité initiale C^{init} (représen-

tant une fraction de la capacité du processeur à fréquence maximale F_{max}). Supposons aussi que la fréquence du processeur est réduite à F_i du fait de la faible charge processeur du système hôte. Le module *NewVMCapacityComputing* est chargé du calcul de la nouvelle capacité C^j à allouer à la VM, de telle en sorte que sa durée d'exécution soit identique à une exécution faite avec une capacité C^{init} et à fréquence F_{max}, c'est-à-dire vérifier que : $T_i^j = T_{max}^{init}$.

D'après l'équation 7.3, $C^j = \frac{T_i^{init} * C^{init}}{T_i^j * \text{cft}_i}$

D'après l'équation 7.2, $T_i^j = \frac{T_{max}^j}{ratio_i * \text{cf}_i}$, donc $T_i^{init} = \frac{T_{max}^{init}}{ratio_i * \text{cf}_i}$

Ainsi, $C^j = \frac{T_{max}^{init} * C^{init}}{ratio_i * \text{cf}_i * T_i^j * \text{cft}_i}$

Nous souhaitons avoir : $T_i^j = T_{max}^{init}$; donc :

donc $C^j = \frac{T_{max}^{init} * C^{init}}{ratio_i * \text{cf}_i * T_{max}^{init}} = \frac{C^{init}}{ratio_i * \text{cf}_i * \text{cft}_i}$

En somme :

$$C^j = \frac{C^{init}}{ratio_i * \text{cf}_i * \text{cft}_i} \tag{7.5}$$

Cette équation de calcul de nouvelle capacité montre qu'il est possible de compenser la perte.

Ainsi,

- si nous exécutons une VM avec une capacité de 20% à la fréquence F_{max}
- si nous réduisons la fréquence du processeur de F_{max} à F_i, par exemple une réduction de moitié de la fréquence maximale telle que le $ratio_i = 0.5$
- nous pouvons modifier la capacité de la VM et lui assigner $20 \div (0.5 * \text{cf}_i * \text{cft}_i)$ dans le but d'avoir la même capacité de calcul. Dans ces conditions, il est possible d'obtenir pour cette VM le même temps de calcul.

7.4.3 Le module *NewVMCapacityAllocation*

Après avoir identifié la nouvelle fréquence du processeur ainsi que les nouvelles capacités de VMs, le module *NewVMCapacityAllocation* assigne à chaque VM sa capacité. Concrètement, il parcourt l'ensemble des VMs démarrées sur le système hôte et modifie (via un accès exclusif aux propriétés de la VM) sa capacité. De manière analogue à l'ordonnanceur par défaut de Xen, notre ordonnanceur de VMs *pas-sla-scheduler* ne rajoute aucun délai lors de l'allocation des capacités des VMs.

7.5 Implantation logicielle de *pas-sla-scheduler*

Pour présenter la mise en œuvre de *pas-sla-scheduler*, nous définissons et exploitons quelques variables contenant des informations descriptives du processeur et des VMs. Nous définissons aussi des informations utiles pour le calcul de la fréquence du processeur et des capacités à attribuer aux VMs. L'ensemble des définitions sont les suivantes :

- *VM[]* représente la liste des VMs démarrées sur le système hôte et *nbVM* le nombre de VMs,
- *Capacite[]* représente le tableau de capacité CPU associée à chaque VM. Il est de la même taille que VM[],
- *Freq[]* est le tableau trié (dans l'ordre croissant des valeurs de fréquence) des différents niveaux de fréquence du processeur. *fmax* représente le nombre de niveau de fréquence et *Freq[fmax]* la fréquence maximale.
- *CF[]* est le tableau des coefficients de proportionnalité cf_i associé à chaque niveau de fréquence
- *CFl* et *CFt* représentent respectivement les coefficients de proportionnalité cfl et cft associés au processeur
- *CurrentFreq* représente la fréquence courante du processeur.
- *VM_load* est la charge de la VM vu de l'OS invité. Par exemple, en considérant nos précédents scénarios, la V20 surchargée possède une VM_load de 100% .
- *VM_global_load* est la contribution de la VM à la charge du processeur. Par exemple, la V20 surchargée (VM_load = 100%) à laquelle était allouée une capacité de 20 contribue à hauteur de 100% de 20% = 20% de charge du processeur. De manière générale, si une capacité *VM_capacite* est allouée à une VM alors sa charge globale est approximativement égale à $VM\_global\_load = VM\_load * VM\_capacite$.
- *Global_load* représente la charge totale du CPU [2]. Elle est égale à la somme de la charge globale de toutes les VMs qui s'exécutent : $Global\_load = \sum VM\_global\_load$.
- *Absolute_load* représente la charge du CPU que nous aurions obtenue s'il fonctionnait à sa fréquence maximale. Sur la base des rapports de proportionnalités précédemment relevés, la charge absolue du CPU est : $Absolute\_load = Global\_load * \frac{CurrentFreq}{Freq[max]} * cfl$.

Algorithme de calcul de fréquence A chaque top horloge de l'ordonnanceur de VMs, nous évaluons la nouvelle fréquence appropriée à la charge absolue *Absolute_load* du processeur, tel que décrit par l'algorithme 4. Dans cet algorithme, un ensemble de calcul et de comparaison sont effectués de manière itérative sur les fréquences existantes sur la plateforme (ligne 5). Conformément à nos suppositions relatives aux fréquences, pour chaque niveau de fréquence existant nous calculons,

2. Notons qu'à chaque fois où il est fait mention de *Global_load*, cette valeur représente une moyenne arithmétique de 3 dernières charges CPU.

le ratio qui lui est associé par rapport à la fréquence maximale (ligne 6). A l'aide de ce ratio, nous calculons ensuite la charge absolue fictive (ligne 7). Nous vérifions ensuite si la charge absolue du CPU (ligne 3) peut entièrement être absorbée par ce niveau de fréquence (ligne 8) via la comparaison avec la charge absolue fictive. Deux situations sont possibles à la fin de cet algorithme : soit il existe une fréquence (inférieure à la fréquence maximale) capable d'absorber la charge absolue du processeur (ligne 9), soit le système est placé en fréquence maximale (ligne 11).

Algorithme 4. Calcul de la nouvelle frequence du CPU

```
1  int computeNewFreq(){
2  int ratio = Current_freq/Freq[fmax];
3  int Absolute_load = Global_load * ratio * \text{cfl};
4
5   for (i=1; i<=fmax; i++) {
6       ratio = Freq[i]/Freq[fmax];
7       int Absolute_load_fictive = 100 * ratio * \text{cfl};
8      if (Absolute_load_fictive > Absolute_load)
9        return Freq[i];
10       }
11  return Freq[fmax];
12  }
```

Algorithme de calcul de capacité A chaque top horloge, la modification éventuelle du niveau de fréquence entraîne une ré-évaluation de la capacité des VMs. Ce calcul et cette allocation sont décrits par l'algorithme 5. Pour chaque nouvelle fréquence, nous calculons le ratio de fréquence qui lui est associée (ligne 3). Pour chaque VM en cours d'exécution sur le système, nous calculons sa nouvelle capacité correspondant à la nouvelle fréquence du processeur (ligne 5) et la lui assignons (ligne 6). La capacité de la VM s'accroit lorsque le fréquence du processeur décroit.

Algorithme 5. Calcul et allocation des capacités VM

```
1 void updateCapacite()
2 {
3   int ratio = newFreq/Freq[fmax];
4   for (vm=1; i<=nbVM; vm++) {
5     int newCapacite = Capacite[vm]/(ratio * CF[newFreq] * \text{cft
        });
6     setCapacite(VM[vm], newCapacite);
7   }
8 }
```

Une remarque importante à relever dans cet algorithme repose sur la somme des capacités allouées aux VMs. Si le niveau de fréquence du processeur est très bas, la somme des capacités allouées aux VMs est supérieure à 100%. Ceci s'explique par le fait que nous calculons la capacité des VMs de façon individuelle et indépendante. En effet, suite à une baisse de fréquence, nous augmentons la capacité de toutes les

VMs, aussi bien (i) de celles qui ont une faible charge CPU et qui n'utiliseront jamais la capacité qu'on leur alloue ; (ii) que celles qui utilisent grandement leur CPU et pourraient être lésées par la baisse de la fréquence.
Si toutefois, la charge CPU des VMs précédemment sous chargées augmente, le système constatera également une augmentation de sa charge CPU globale. Cette augmentation de la charge globale du processeur conduira à une augmentation de la fréquence du processeur. Par ricochet, la capacité des VMs, précédemment sous chargées, diminuera pour converger vers leur capacité initialement allouée.

7.6 Exploitation

Dans sa version actuelle, notre prototype *pas-sla-scheduler* est utilisable dans le contexte suivant : (1) avoir une architecture virtualisée sous Xen ; (2) déterminer pour chaque famille de processeurs de l'infrastructure le coefficient de proportionnalité de chaque niveau de fréquence. Au démarrage du système, l'ordonnanceur *pas-sla-scheduler* peut être spécifié en ligne de commande ou dans le fichier de démarrage (grub.conf). Un fichier contenant les niveaux de fréquence et les coefficients correspondants peut également être fournis en option et exploité par *pas-sla-scheduler* lors de l'initiation du système hôte.

Notre prototype a certes été implanté sous Xen, toutefois l'ensemble les exigences de conception d'un ordonnanceur de VMs de type PAS restent les mêmes. L'implantation de *pas-sla-scheduler* dans d'autres hyperviseurs est donc facilement envisageable.

7.7 Synthèse

Dans ce chapitre, nous avons présenté les 2 principaux services qui constituent *pas-sla-scheduler*. Pour chacun d'eux, nous avons présenté les modules internes, leur fonctionnement et leur interaction. A cette description fonctionnelle, nous avons présenté quelques algorithmes que nous avons implanté afin d'atteindre l'objectif initial voulu par *pas-sla-scheduler*.

Le chapitre suivant porte sur l'évaluation et la validation de notre prototype.

Chapitre 8

Validation

Contents

Ce chapitre porte sur l'évaluation et la validation de notre contribution. Il est organisé en 4 parties :

1. la description de l'environnement matériel et logiciel sur lequel se base l'ensemble de nos expériences ;
2. la vérification des hypothèses de proportionnalités que nous avons définies en section 7.1 ;
3. la validation du dysfonctionnement fonctionnel identifié entre l'ordonnanceur de VMs et le DVFS. Elle se rapporte à montrer ce dysfonctionnement dans le cas du DVFS utilisé avec les ordonnanceurs à capacité fixe et à capacité variable présents dans Xen ;
4. la validation de notre prototype. Il s'agit de reprendre les expériences effectuées lors des tests précédents en utilisant notre prototype.

Pour chacune des expériences réalisées, nous décrivons le contexte d'évaluation ainsi que le scénario de notre expérience. A la suite, nous présentons et analysons les résultats obtenus en fonction des conditions d'expérimentation.

8.1 Environnement matériel et logiciel

Nos expériences ont été réalisées sur un nœud DELL Optiplex 755, avec un processeur Intel Core 2 Duo 2,66 avec 4Go de RAM. Sur cette machine, nous avons installé la distribution Debian squeeze de Linux (avec le noyau 2.6.32.27) comme système d'exploitation. Nous l'avons ensuite configuré pour un fonctionnement en monocoeur. L'hyperviseur Xen (dans sa version 4.1.2) a été utilisé comme solution de virtualisation dans l'ensemble de nos évaluations.
En fonction des objectifs visés par nos évaluations, les expériences décrites dans ce chapitre sont réalisées grâce à deux applications :

- lorsque l'objectif est de mesurer un temps d'exécution, nous utilisons une application qui calcule une approximation de π. Cette application est appelée π-*app* dans la suite,
- lorsque l'objectif est de mesurer une charge processeur, nous utilisons une application web (un serveur CMS Joomla) qui reçoit une charge (des requêtes http) générée par httperf [94]. Le serveur web se compose de Joomla 1.7, dans lequel est intégré Apache 2.2.16, PHP 5.3.3, une base de données MySQL 1.5.49 et modphp5. Cette application est appelée *Web-app* dans la suite.

Une seconde machine de caractéristiques identiques est utilisée comme client lors de l'injection de la charge de travail. Les 2 machines sont connectées via un switch DELL Powerconnect 2708. Les VMs ont les mêmes configurations matérielles.

8.2 Vérification des hypothèses

Dans cette section, nous vérifions les rapports de proportionnalité que nous avons introduits en section 7.1. Nous vérifions également que le calcul des coefficients est fonction des architectures de processeur.

8.2.1 Vérification des proportionnalités

Comme mentionné en section 6.1, la mise en œuvre de notre ordonnanceur de VMs (*pas-sla-scheduler*) repose sur 2 hypothèses principales : la proportionnalité de la fréquence et de la performance (équation 7.1 et 7.2) et la proportionnalité de la capacité et de la performance (équation 7.3).
Afin de valider ces deux hypothèses, nous avons effectué les expériences suivantes sur un système virtualisé utilisant l'ordonnanceur à capacité fixe (credit scheduler) de Xen :

- **Proportionnalité entre fréquence et performance.** Nous avons exécuté différents profils de charge de travail sur web-app à différentes fréquences de

processeur. Pour chacun des profils, nous avons mesuré la charge L_i du processeur aux différentes fréquences de processeur F_i existantes. Nous avons ensuite calculé les ratios $\frac{L_{max}}{L_i}$ *et* $\frac{F_i}{F_{max}}$ pour chaque charge de travail afin de déterminer les coefficients cfl_i liés à chaque niveau de fréquence du processeur. Ce calcul a permis de vérifier que *pour chaque fréquence, ces coefficients sont constants pour les divers profils de charge de travail* (validant ainsi équation 7.1).

Nous avons effectué le même test avec l'application π-app. Nous avons exécuté π-app sur un processeur à différentes fréquences et avec différentes valeurs de capacité de VM. Pour chacun des calculs, nous avons mesuré leurs temps d'exécution. Le calcul du coefficient de proportionnalité s'est révélé non constant pour chaque niveau de fréquence. Afin de prendre en compte cette particularité, nous avons opté pour une discrétisation des coefficients. Pour chaque niveau de fréquence, nous lui avons allouée le coefficient cf_i correspondant. Ceci nous a permis de vérifier la proportionnalité entre le ratio de fréquence et le ratio de temps d'exécution (équation 7.2).

– **Proportionnalité entre capacité et performance.** Nous avons exécuté π-app sur une VM configurée avec différents valeurs de capacité de CPU. Nous avons considéré la première valeur de capacité allouée comme notre capacité initiale. Pour chaque calcul à capacité (indice j) donnée, nous avons mesuré le temps d'exécution ; calculé le ratio de capacité $(\frac{C^j}{C^{init}})$ et le ratio de temps d'exécution $(\frac{T_i^{init}}{T_i^j})$. Nous avons obtenu un coefficient constant pour chaque niveau de fréquence. Nous avons de cette manière vérifié l'équation 7.3.

Afin de vérifier l'exactitude de nos rapports de proportionnalité, nous avons effectué une expérience afin de valider l'équation 7.5 utilisée pour le calcul de la nouvelle capacité de CPU des VMs. Nous avons exécuté π-app à fréquence maximale (2667 MHz) avec différents capacité initiales (10, 20, 30 ...). Ensuite, nous avons effectué la même expérience à une fréquence donnée 2133 MHz. Dans cette seconde série de calcul, la capacité allouée à chaque VM correspond à la capacité calculée sur la base de la capacité initiale grâce à l'équation 7.5. Ces capacités représentent les capacités associées aux capacités initiales (10, 20, 30,...) qui compensent l'effet de la baisse de fréquence.

La figure 8.1 montre les résultats obtenus à la suite de cette expérience. L'axe des X en bas représente les capacités initiales de la VM et l'axe X dans la partie supérieure les capacités calculées. L'axe Y correspond à la durée d'exécution. Pour chaque capacité, la courbe représente le temps d'exécution de l'application à la fréquence concernée. La superposition (presque parfaite) des 2 courbes montre qu'en compensant la capacité de CPU allouée à une VM, il est possible de conserver la même durée d'exécution. Toutefois, pour les valeurs de capacité très grande, la baisse de la fréquence conduit à des valeurs de capacité supérieure à 100. Ces dernières ne pouvant être mise en œuvre, il est possible d'observer quelques pertes de performance

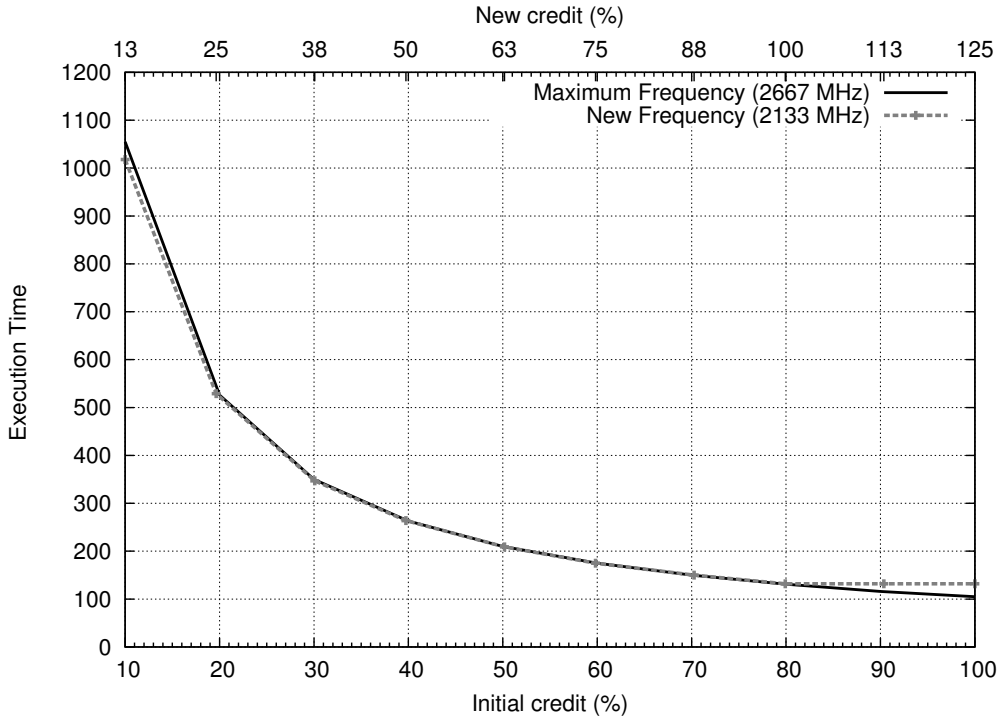

Fig 8.1. Compensation de la baisse de fréquence par l'allocation de capacité

(*cas de la capacité initiale à 100 sur la figure*) si la VM passe en mode "trashing". Cette expérimentation montre que nous pouvons effectivement compenser une réduction de la fréquence avec une augmentation proportionnelle de la capacité des VMs.

8.2.2 Vérification sur d'autres architectures

Afin de confirmer nos hypothèses de proportionnalité, nous avons vérifié leur validité sur d'autres architectures. Par conséquent, nous avons mesuré pour différentes charges de travail et pour chaque fréquence de processeur le coefficient cf_i pour différents types de machines disponibles sur Grid5000 (la grille nationale de calcul d'expérimentation). Dans le tableau 8.1, nous présentons uniquement les valeurs cf_i obtenues pour les fréquences minimales.

	Intel Xeon X3440	**Intel Xeon L5420**	**Intel Xeon E5-2620**	**AMD Opteron 6164 HE**	**Intel Core i7-3770**
cf_{min}	0,94867	0,99903	0,80338	0,99508	0,86206

TABLE 8.1. Valeur de cf_{min} pour différentes architectures

Bien que le coefficient cf_{min} soit pour la plupart du temps égal à un, nous avons constaté qu'il peut varier de manière significative sur les architectures particulières (par exemple Intel Xeon E5-2620).

8.3 Vérification des incompatibilités entre l'ordonnanceur et le DVFS

8.3.1 Profil d'exécution

Dans l'ensemble de cette section de vérification, nous nous appuyons sur l'application Web-app décrite auparavant. Nous considérons deux machines virtuelles appelées V20 et V70, avec respectivement 20% et 70% de capacité de CPU allouée initialement. Les 10% restants de capacité sont destinées à l'hyperviseur (le dom0 Xen).

Afin de relever le problème d'incompatibilité dans le fonctionnement indépendant du DVFS et de l'ordonnanceur de VMs, nous avons défini un profil de charge des deux machines virtuelles subdivisé en 3 phases : ***inactif - actif - inactif*** :

- ***Inactif*** : au cours de cette phase, la VM ne reçoit pas de charge de la part de l'injecteur (httperf).
- ***Active*** : pendant cette phase, la machine virtuelle peut recevoir deux types de charge : (1) soit l'injecteur est configuré pour générer une charge qui représente *exactement* 100% de la capacité de la VM mais pas plus (nous parlons de *exact load*), (2) soit il est configuré pour générer une charge *supérieure* à la capacité de la VM (nous parlons de *trashing load*).

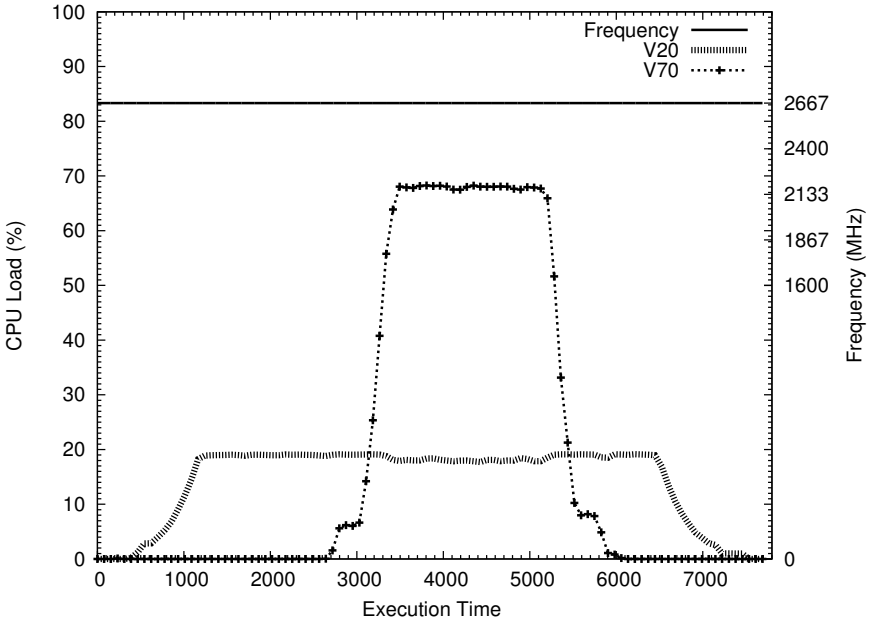

Fig 8.2. Profil de charge (à fréquence maximale)

La figure 8.2 montre la charge globale des machines virtuelles (VM_global_load) lors de l'exécution de ce profil de charge avec l'ordonnanceur à capacité fixe (sched-credit) de Xen, et à la fréquence maximale du processeur (la fréquence est indiquée sur l'axe Y sur le côté droit). Tel que défini dans la section 7.5, la charge

VM_global_load représente la contribution de la VM à la charge du processeur. Notons ici que ce profil de charge est identique quelque soit le type de charge injecté (exact load ou trashing load) ; car la configuration de la capacité d'une VM avec l'ordonnanceur "sched-credit" de Xen indique le pourcentage maximal de capacité que peut avoir une VM. Cette figure représente le profil d'exécution que nous utiliserons dans le reste de nos évaluations.

8.3.2 Incompatibilité : ordonnanceur de VMs et DVFS

Dans la présentation du problème d'incompatibilités entre le DVFS et l'ordonnanceur de VMs, nous nous sommes intéressés à 2 familles d'ordonnanceurs. Par chacune d'elle, nous vérifions le constat présenté en section 4.2.3.
Xen dispose de plusieurs ordonnanceurs. L'ordonnanceur *sched-credit* est un ordonnanceur à capacité fixe et *SEDF* un ordonnanceur à capacité variable. Pour mettre en exergue les problèmes que posent ces différents ordonnanceurs, nous les étudions dans différents scénarios.

Scénario 1 : Ordonnanceur à capacité fixe (sched-credit) Dans ce scénario, nous injections aux VMs une charge exacte correspondant à leur capacité conformément au profil d'exécution présenté en section 8.2. Le système virtualisé est configuré de façon à utiliser l'ordonnanceur à capacité fixe et le governor Ondemand. Le monitoring de la charge CPU des VMs et de la fréquence courante du processeur, nous a permis d'obtenir les graphes ci-dessous.

En analysant la figure 8.3, nous observons que le governor par défaut (Ondemand) est assez agressif et instable. Nous l'avons configuré de façon à le rendre moins agressif et plus stable comme l'illustre la figure 8.4. Pour plus de lisibilité sur l'ensemble des figures, la suite de nos évaluations utilisent le governor ondemand configuré pour éviter les oscillations.

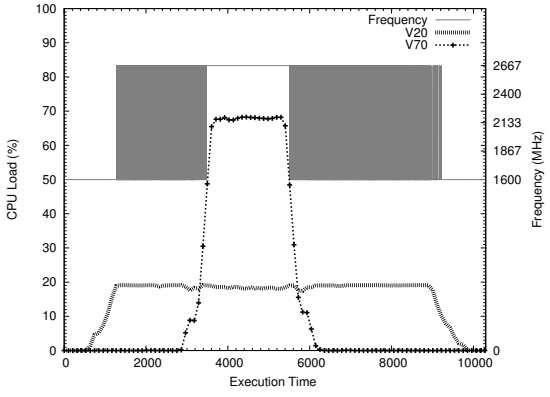

Fig 8.3. Charge globale avec le governor Ondemand / ordonnanceur Credit / charge exacte

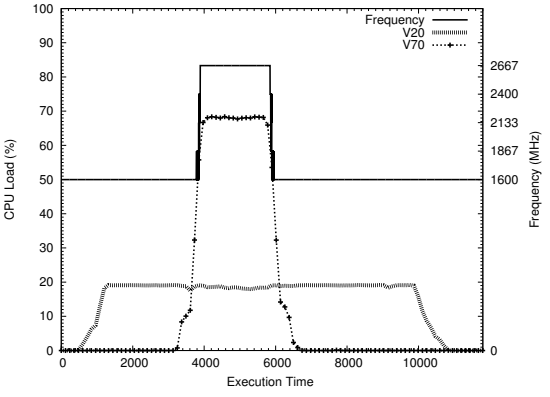

Fig 8.4. Charge globale avec notre governor Ondemand / ordonnanceur Credit / charge exacte

Lorsque les VMs sont en phase active, leur charge globale est de 70% pour la V70 et 20% pour la V20. Cette charge globale représente leur contribution à la charge du processeur. Tandis que les 2 premières figures précédentes montrent la charge globale des VMs, la figure 8.5 quant à elle, montre la charge absolue (définie en section 7.5 comme la charge du processeur correspondant à un processeur exécuté à sa fréquence maximale ou plus précisément Absolute_load = Global_load * $\frac{CurrentFreq}{Freq[max]} * cfl$). Nous observons qu'au cours de la première phase d'exécution (lorsque la V20 est active et la V70 inactive), la charge absolue de la V20 est proche de 10%. Ceci est dû à la baisse de la fréquence du processeur parce qu'il est globalement chargé à 20%. Cependant, lorsque la V70 devient active, elle génère une charge globale permettant d'accroître la fréquence du processeur au point d'atteindre son maximum. A ce moment, la V20 reçoit toute la capacité de CPU souscrite et atteint une charge absolue de 20%. En somme, la V20 ne peut recevoir sa capacité totale que lorsque le processeur est à sa fréquence maximale. Dans les autres cas, elle subit une dégradation de performance.

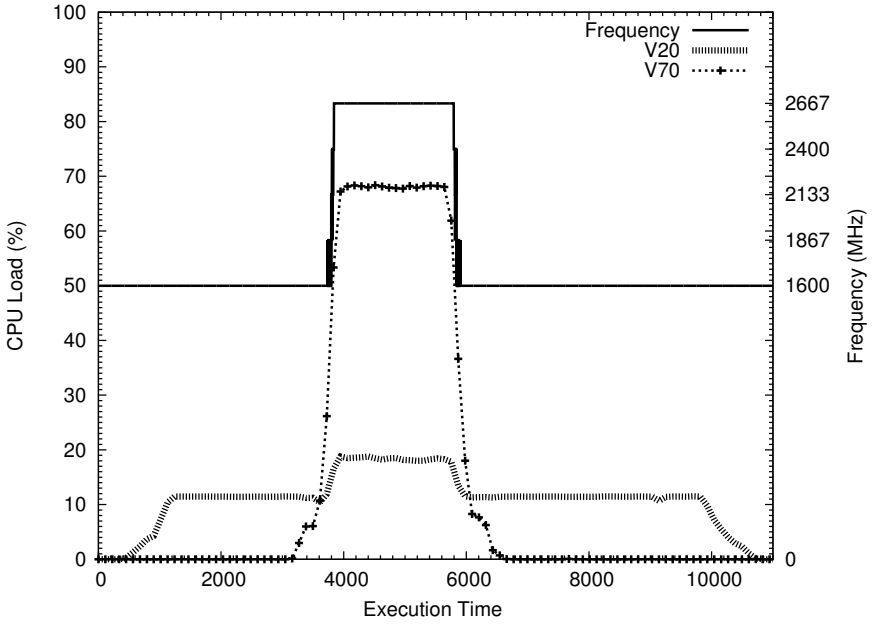

Fig 8.5. Charge absolue avec le governor Ondemand / ordonnanceur Credit / charge exacte

L'ordonnanceur à capacité variable résout-il ce problème ? Nous avons effectué la même expérience en utilisant l'ordonnanceur à capacité variable (SEDF) de Xen. Rappelons qu'avec SEDF, la capacité de temps CPU non utilisée par une VM est distribuée aux VMs actives qui en ont besoin.
Par conséquent, en observant la figure 8.6, durant la première phase (lorsque la V20 est active et la V70 inactive), la V20 a une charge globale supérieure au 20% souscrit. Ceci est dû au fait que les tranches de temps CPU non utilisées par la V70 lui ont été allouées. Lorsque la V70 devient active, les capacités initiales sont respectées et la V20 obtient exactement 20% de charge globale (processeur à fréquence maximale).

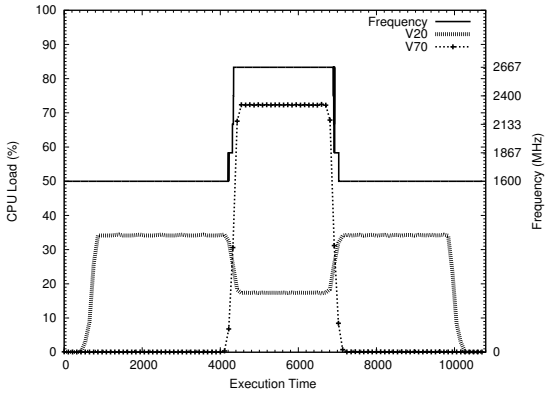

Fig 8.6. Charge globale avec notre governor ondemand/ ordonnanceur SEDF / charge exacte

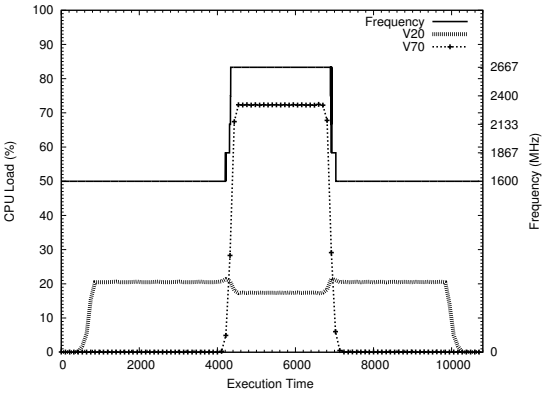

Fig 8.7. Charge absolute avec notre governor/ ordonnanceur SEDF / charge exacte

En analysant la figure 8.7 représentant les charges absolues des VMs lors de cette expérience, nous constatons que les tranches de temps non utilisées allouées à la V20 ont permis de compenser la pénalité qu'engendre la baisse de fréquence. la V20 obtient 20% de charge absolue durant toute l'expérience. On pourrait ainsi dire que SEDF apporte une solution au fait qu'une VM active peut être victime de la baisse de fréquence si les autres sont inactives.

Scénario 2 : Ordonnanceur à capacité variable (SEDF) Toutefois, l'ordonnanceur à capacité variable SEDF ne résout pas réellement le problème que nous avons identifié. Dans nos expériences précédentes, nous avons utilisé des charges exactes (qui représente exactement 100% de la capacité de la VM). Si nous injectons une charge qui excède la capacité de la VM (trashing load), nous observons (Figure 8.8) que durant la première phase (lorsque la V20 est active et la V70 inactive), SEDF alloue l'ensemble des tranches de temps processeur non utilisées à la V20.
A terme, cette situation conduit à conserver la fréquence du processeur à sa valeur maximale. Durant la première phase, la consommation de la V20 atteint 85% du processeur. Ceci n'est pas acceptable du point de vue du fournisseur parce qu'il ne tire aucun profit du fait de l'inactivité de la V70 et la V20 utilise une capacité pour laquelle elle n'a pas souscrit. Cette situation peut être considérée, du point de vue du fournisseur, comme du gaspillage de ressources parce qu'empêchant toute possibilité de consolidation. Durant la seconde phase, lorsque la V70 devient active, SEDF garantit les capacités souscrites par chacune des VMs.
Notons ici que les figures décrivant les charges globale et absolue sont identiques puisque le processeur reste à sa fréquence maximale durant toute l'expérience. Nous n'avons donc présenté qu'une seule figure.

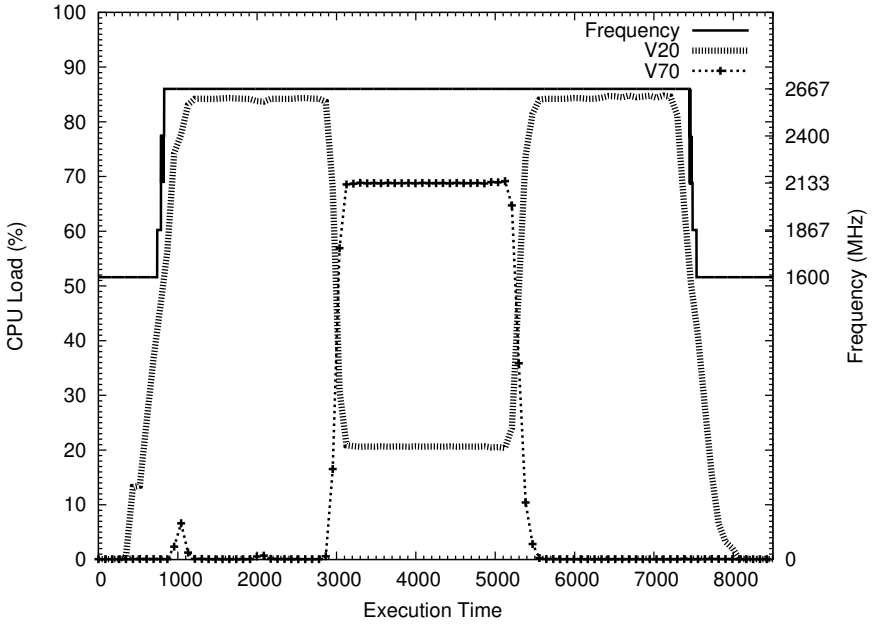

Fig 8.8. charge globale ou absolute avec notre governor / ordonnanceur SEDF / thrashing load

8.4 Validation de notre approche

Dans son fonctionnement, notre prototype *pas-sla-scheduler* recalcule les nouvelles capacités de CPU à allouer aux VMs en fonction de la fréquence courante du processeur. En conséquence, il offre les mêmes atouts que SEDF dans le cas où les VMs ont une charge exacte. En plus, l'ordonnanceur *pas-sla-scheduler* apporte la possibilité de toujours garantir le respect de la capacité de CPU souscrit même pour les VMs qui génèrent une charge de type *trashing load*.

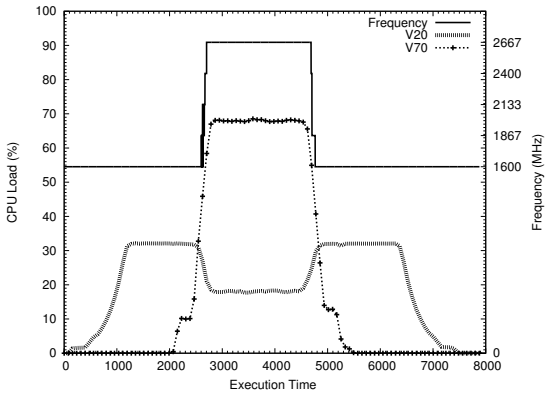

Fig 8.9. Charge globale avec notre ordonnanceur/ thrashing load

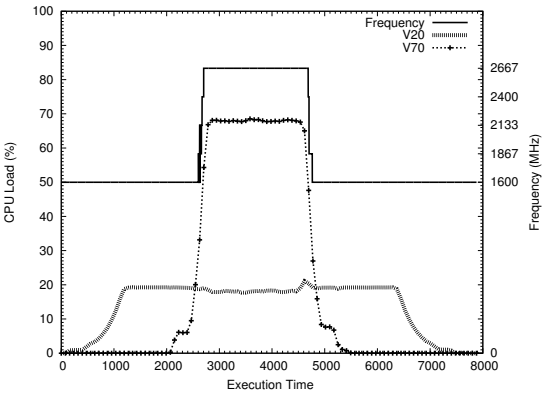

Fig 8.10. Charge absolute avec notre ordonnanceur/ thrashing load

Dans la figure 8.9, nous observons qu'une capacité supérieure à la capacité souscrit est allouée à la V20 pendant la première phase (la V20 est active et la V70 inactive). Cette valeur est calculée par l'ordonnanceur *pas-sla-scheduler* dans le but de compenser la baisse de fréquence du processeur (1600 MHz). Durant la seconde

phase, la V20 reçoit exactement 20% de capacité vu que le processeur est à sa fréquence maximale. Grâce à cette stratégie de calcul et de compensation, la charge absolue de chacune des VMs correspond aux capacités initialement souscrites (Figure 8.10).

8.5 Synthèse

Tout au long de cette partie, il a été question de proposer un prototype d'ordonnanceur de VMs permettant de respecter le SLA des VMs tout en assurant une utilisation optimale des ressources. En effet, grâce à la possibilité de changer la vitesse d'exécution des périphériques, nous avons défini un modèle de gestion d'énergie. Ce modèle consiste à faire varier dynamiquement la vitesse d'exécution d'un périphérique (soit grâce à une fonctionnalité intégrée dans le périphérique soit par un regroupement des traitements du périphérique) de façon à réduire sa consommation énergétique. Cette variation de vitesse a comme effet de bord d'impacter les performances des VMs qui l'utilisent. Cette situation a fait l'objet d'une étude, au terme de laquelle, nous recommandons de concevoir des gestionnaires de ressources capables de prendre en compte la vitesse d'exécution de la ressource gérée.
Spécialement dans le cas du CPU, nous avons dans cette partie proposé un ordonnanceur de VMs nommé *pas-sla-scheduleur* qui en fonction de la vitesse d'exécution du processeur, adapte proportionnellement les capacités à allouer aux VMs. Dans ce chapitre spécifiquement, nous nous sommes concentrés sur la validation de notre approche. Nous avons initialement défini l'environnement matériel et logiciel de nos évaluations. Nous avons ensuite montré le mécanisme de calcul des différents coefficients de proportionnalités et les avons validés via une expérience de compensation. Nous avons ensuite identifié 2 ordonnanceurs de Xen, entrant chacun dans l'une des catégories d'ordonnanceurs (capacité variable ou fixe) présentés, et pour chacun d'eux, nous avons montré qu'ils ne remplissent pas les critères que nous avons défini en section 5.1. Enfin, nous avons montré comment notre prototype permettait de résoudre ce problème et était conforme à nos critères.

Le chapitre suivant porte sur la conclusion et présente les différentes perspectives que nous avons à la suite de ces travaux.

Chapitre 9

Conclusion et perspectives

Contents

9.1 Conclusion

L'expansion du cloud computing, ces dernières années, trouve sa principale justification dans le coût élevé de la gestion locale des infrastructures informatiques. En effet, le coût matériel, logiciel, humain et énergétique d'une infrastructure informatique est fonction du nombre d'équipements qu'elle contient. Plus les serveurs sont déployés avec redondance et/ou l'allocation des ressources surdimensionnée (entrainant ainsi un gaspillage de ressources), plus le coût de l'infrastructure sera élevé. De nombreuses entreprises ont donc entrepris de déléguer la gestion de leur infrastructure à des fournisseurs de services. Ces derniers se caractérisent, entre autre, par le modèle de service qu'ils offrent et la stratégie d'allocation de ressource à la demande afn d'éviter tout sur/sous dimensionnement. Quel que soit le service offert, le fournisseur convient avec le client d'un accord de qualité de service (SLA - Service layer Agreement) qu'il est tenu de respecter.

Dans ce contexte, le problème qui se pose est de réduire les coûts de fonctionnement pour le fournisseur tout en respectant le SLA des clients. Pour résoudre ce problème de coût de fonctionnement et particulièrement du coût énergétique, des politiques de gestion d'énergie ont été proposées. L'étude de quelques solutions de gestion d'énergie que nous avons effectuée fait ressortir une similitude entre elles : ces solutions sont basées sur le changement de vitesse d'exécution des périphériques. En effet, grâce aux changements d'état des périphériques (changement qui se traduit

par une variation de vitesse d'exécution), l'application d'une politique de gestion d'énergie sur un périphérique consiste à faire varier sa vitesse en fonction de sa charge. En fonction du périphérique concerné, cette variation de vitesse est effectuée de façon native, grâce à des fonctionnalités intégrées dans le périphérique, ou par simulation via des regroupements (temporel et spatial) des traitements qu'effectuent le périphérique.

L'utilisation de ces politiques de gestion d'énergie dans un système virtualisé peut entrainer un non respect du SLA souscrit par le client. Nos expériences avec des ordonnanceurs de CPU montrent qu'en fonction du gestionnaire de ressources utilisé, soit la VM ne reçoit pas le SLA souhaité du fait de la baisse de la vitesse d'exécution du périphérique, soit elle utilisera bien plus de ressources entrainant ainsi un gaspillage du point de vue du fournisseur. Notre contribution dans ce contexte a été de fournir un prototype de gestionnaire de CPU (ordonnanceur de VMs) capable de garantir le respect du SLA tout en favorisant la baisse de la consommation énergétique.

Dans le cadre de cette thèse, nous avons proposé les principes généraux utiles pour la mise en œuvre d'un ordonnanceur de VMs respectant 2 critères : le respect du SLA et l'efficience énergétique. Nous avons ensuite appliqué ces principes pour proposer notre ordonnanceur *pas-sla-scheduler*. Cet ordonnanceur monitore la charge courante du CPU et ajuste la fréquence d'exécution du CPU en conséquence. Conscient du fait qu'une modification de fréquence de CPU peut influencer le SLA des VMs qui l'utilisent, notre ordonnanceur *pas-sla-scheduler* ajuste dynamiquement la capacité allouée au VMs. Cet ajustement consiste à augmenter (respectivement diminuer) la capacité des VMs lorsque la fréquence du processeur est abaissée (respectivement augmentée). Cette nouvelle capacité (liée à la nouvelle fréquence) est équivalente à la capacité initiale liée à la fréquence maximale.

Pour finir, nous avons évalué notre prototype à plusieurs niveaux. Nous avons commencé par vérifier la validité des hypothèses de proportionnalité que nous avons définies entre la fréquence du CPU, la performance des VMs (charge CPU et durée d'exécution des applications) et la capacité initiale allouée aux VMs. Cette validation a permis de valider les formules de calcul utilisées lors de la recherche de la nouvelle fréquence du CPU ou lors du calcul de la nouvelle valeur de capacité de VMs. Enfin, nous avons comparé notre prototype à différents ordonnanceurs dans différents systèmes de virtualisation. Ces évaluations montrent que contrairement aux autres systèmes, notre prototype respecte le SLA de chaque VM sans permettre aux VMs de surconsommer (et gaspiller les ressources du fournisseur).

9.2 Perspectives

Tout au long de la réalisation de cette thèse, nous avons identifié différents axes potentiels de prolongement de nos travaux. Ils peuvent être classés en deux caté-

gories : les travaux réalisables dans un futur proche en se basant sur le modèle de gestion d'énergie que nous avons défini, et ceux réalisables dans un futur plus lointain concernant de nouvelles pistes de recherche. Nous présentons ces perspectives dans les sections suivantes

9.2.1 Perpectives à court terme

Validation en condition réelle Les scénarios de vérification et de validation que nous avons effectués sont basés sur des données de charge synthétique sous la forme de *proof Of concept*. Nous envisageons de valider notre approche en nous servant des données réelles d'un fournisseur de services. A ce effet, nous exploiterons les données représentant les traces d'exécution réelles de VMs d'un fournisseur (*Eolas Datacenter* [1]) avec qui nous collaborons à travers un projet ANR (ctrl-green). Le but est d'analyser ces données et de construire une simulation de scénario basée sur ces traces [2] afin de valider notre approche. De plus, nous souhaitons effectuer une évaluation du gain énergétique éventuel que pourrait apporter (ou pas) l'utilisation de notre approche.

Application du modèle de gestion d'énergie Lors de la définition de notre modèle de gestion d'énergie, nous avons affirmé qu'il est applicable sur tout type de périphérique. L'une de nos perspectives à court terme consiste à appliquer l'ensemble des principes généraux de conception de gestionnaire de ressources pour d'autres types périphériques (disque , RAM, carte réseau).

9.2.2 Perpectives à long terme

Dans le cadre de cette thèse, nous nous sommes focalisés sur des architectures monocoeurs. A long terme, nous souhaitons considérer les spécificités de toutes architectures existantes (multicoeurs, NUMA, multithread, asymétrique, symétrique, etc..) et les utiliser pour étoffer l'ensemble des principes que nous avons présenté dans ce document. Sur la base de ces éléments, nous souhaitons mettre à jour notre ordonnanceur et vérifier si (i) les hypothèses énoncées demeurent valides et, (ii) notre approche permet toujours le respect du SLA tout en permettant une utilisation optimale de l'énergie consommée par le CPU.

L'exécution d'une VM sur un processeur à vitesse variable peut être assimilée à un cluster de machines hétérogènes (notamment de vitesse maximale de processeur différente) à vitesse d'exécution fixe, dans lequel la VM est migrée d'un nœud à un

1. Site : www.businessdecision-eolas.com
2. En anglais : trace based simulation

autre en fonction de sa charge et de la vitesse du nœud. Nous envisageons donc d'implanter les principes généraux définis dans un contexte distribué.

Bibliographie

[Amazon] Amazon. Amazon elastic compute cloud.

[2] Ananthanarayanan, G. and Katz, R. H. (2008). Greening the switch. In *HotPower*.

[3] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Commununication of the ACM*.

[4] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds : A berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley.

[5] Baek, J., Safavi-Naini, R., and Susilo, W. (2008). Public key encryption with keyword search revisited. In *Computational Science and Its Applications*.

[6] Bartholomew, D. (2006). Qemu a multihost multitarget emulator. *Linux Journal*, 2006.

[7] Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*.

[8] Beloglazov, A. and Buyya, R. (2010). Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*.

[9] Benini, L. and de Micheli, G. (2000). System-level power optimization : techniques and tools. *ACM Transactions on Design Automation of Electronic Systems*.

[10] Bianchini, R. and Rajamony, R. (2004). Power and energy management for server systems. *Computer*.

[11] Boneh, D., Di Crescenzo, G., Ostrovsky, R., and Persiano, G. (2004). Public key encryption with keyword search. In *Advances in Cryptology-Eurocrypt 2004*.

[12] Braham, P., Dragovic, B., Fraser, K., Hand, S., Haris, T., Alex, Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. *Proceedings of the nineteenth ACM symposium on Operating systems principles*.

[13] Burd, T. D. and Brodersen, R. W. (2000). Design issues for dynamic voltage scaling. In *Proceedings of the 2000 international symposium on Low power electronics and design*.

[14] Buyya, R., Broberg, J., and Goscinski, A. M. (2010). *Cloud computing : Principles and paradigms*.

[15] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms : Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*.

[16] Carrera, E. V., Pinheiro, E., and Bianchini, R. (2003). Conserving disk energy in network servers. In *Proceedings of the 17th annual international conference on Supercomputing*.

[17] Catthoor, F., Greef, E. d., and Suytack, S. (1998). *Custom memory management methodology : Exploration of memory organisation for embedded multimedia system design*.

[18] Chen, G., Malkowski, K., Kandemir, M., and Raghavan, P. (2005a). Reducing power with performance constraints for parallel sparse applications. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*.

[19] Chen, X., Srivastava, A., and Sorenson, P. (2011). Toward a qos-focused saas evaluation model. *Cloud Computing and Software Services*.

[20] Cherkasova, L., Gupta, D., and Vahdat, A. (2007). Comparison of the three cpu schedulers in xen. *SIGMETRICS Performance Evaluation Review*.

[21] Clark, B., Deshane, T., Dow, E., Evanchik, S., Finlayson, M., Herne, J., and Matthews, J. N. (2004). Xen and the art of repeated research. In *USENIX Annual Technical Conference, FREENIX Track*.

[22] Clark, C., Fraser, K., ans Jacob Gorm Hansen, S. H., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. *USENIX Association Berkeley, CA, USA*.

[23] Colarelli, D. and Grunwald, D. (2002). Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*.

[24] Cully, B. and Warfield, A. (2007). Virtual machine checkpointing. *Xen summit*.

[25] David, H., Fallin, C., Gorbatov, E., Hanebutte, U. R., and Mutlu, O. (2011). Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing*.

[26] Delaluz, V., Kandemir, M., Vijaykrishnan, N., Sivasubramaniam, A., and Irwin, M. J. (2001). Hardware and software techniques for controlling dram power modes. *Computers, IEEE Transactions on*.

[27] Delaluz, V., Sivasubramaniam, A., Kandemir, M., Vijaykrishnan, N., and Irwin, M. J. (2002). Scheduler-based dram energy management. In *Proceedings of the 39th annual Design Automation Conference*.

[28] Deng, Q., Meisner, D., Bhattacharjee, A., Wenisch, T. F., and Bianchini, R. (2012). Multiscale : memory system dvfs with multiple memory controllers. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*.

[29] Deng, Q., Meisner, D., Ramos, L., Wenisch, T. F., and Bianchini, R. (2011). Memscale : active low-power modes for main memory. *ACM SIGPLAN Notices*.

[30] Douglis, F., Krishnan, P., and Marsh, B. (1994). Thwarting the power-hungry disk. In *Proceedings of the 1994 Winter USENIX Conference*.

[31] Dunlap, G. W. (2009). Scheduler development update. *Xen Summit Asia*.

[32] Elnozahy, M., Kistler, M., and Rajamony, R. (2003). Energy conservation policies for web servers. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*.

[33] Fan, X., Ellis, C., and Lebeck, A. (2001). Memory controller policies for dram power management. In *Proceedings of the 2001 international symposium on Low power electronics and design*.

[34] Ferrero, B., Laudebat, R., and Icart, S. (2006). Compromis dvfs/modes faible consommation, gestion mémoire c. belleudy.

[35] Fung, H. T. (2009). System, method, and architecture for dynamic server power management and dynamic workload management for multiserver environment.

[36] Ganesh, L., Weatherspoon, H., Balakrishnan, M., and Birman, K. (2007). Optimizing power consumption in large scale storage systems. *Proceedings of the 11th USENIX workshop on Hot topics in Operating Systems*.

[37] Ganesh, L., Weatherspoon, H., Marian, T., and Birman, K. (2013). Integrated approach to data center power management. *Computers, IEEE Transactions on*.

[Google] Google. Google app engine.

[39] Goth, G. (2007). Virtualization : Old technology offers huge new potential. *IEEE Distributed Systems Online*.

[40] Gunaratne, C. and Christensen, K. (2006). Ethernet adaptive link rate : System design and performance evaluation. In *Proceedings 31st IEEE Conference on Local Computer Networks*.

[41] Gunaratne, C., Christensen, K., Nordman, B., and Suen, S. (2008). Reducing the energy consumption of ethernet with adaptive link rate (alr). *Computers, IEEE Transactions on*.

[42] Gunaratne, C., Christensen, K., and Suen, S. (2006). Ngl02-2 : Ethernet adaptive link rate (alr) : Analysis of a buffer threshold policy. In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*.

[43] Gupta, M., Grover, S., and Singh, S. (2004). A feasibility study for power management in lan switches. In *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*.

[44] Gurumurthi, S., Sivasubramaniam, A., and Natarajan, V. K. (2005). *Disk drive roadmap from the thermal perspective : A case for dynamic thermal management*.

[45] Gurumurthi, S., Zhang, J., Sivasubramaniam, A., Kandemir, M., Franke, H., Vijaykrishnan, N., and Irwin, M. J. (2003). Interplay of energy and performance for disk arrays running transaction processing workloads. In *IEEE International Symposium on Performance Analysis of Systems and Software*.

[46] Habib, I. (2008). Virtualization with kvm. *Linux Journal*.

[47] Hampel, C. E., Barth, R. M., Davis, P. G., May, B. A., Satagopan, R., and Ware, F. A. (2001). Rambus dram (rdram) apparatus and method for performing refresh operations.

[48] Hiremane, R. (2007). Intel virtualization technology for directed i/o (intel vt-d). *Technology@ Intel Magazine*, 4.

[49] Hsu, C.-H., Kremer, U., and Hsiao, M. (2001). Compiler-directed dynamic frequency and voltage scheduling. In *Power-Aware Computer Systems*.

[50] Hsu, S.-T., Yang, C.-C., and Hwang, M.-S. (2013). A study of public key encryption with keyword search. *International Journal of Network Security*.

[51] Hu, J., Gu, J., Sun, G., and Zhao, T. (2010). A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*.

[52] Hu, X., Schmid, S., Richa, A., and Feldmann, A. (2013). Optimal migration contracts in virtual networks : Pay-as-you-come vs pay-as-you-go pricing. In *Distributed Computing and Networking*. Springer.

[53] Huang, H., Shin, K. G., Lefurgy, C., and Keller, T. (2005). Improving energy efficiency by making dram less randomly accessed. In *Proceedings of the 2005 international symposium on Low power electronics and design*.

[54] Huang, S. and Feng, W. (2009). Energy-efficient cluster computing via accurate workload characterization. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*.

[55] Hurwitz, J., Bloor, R., Kaufman, M., and Halper, F. (2009). *Cloud computing for dummies*.

[56] Inc., R. (1999). Rambus dram.

[57] Jang, J.-W., Jeon, M., Kim, H.-S., Jo, H., Kim, J.-S., and Maeng, S. (2011). Energy reduction in consolidated servers through memory-aware virtual machine scheduling. *Computers, IEEE Transactions on*.

[58] Jardosh, A. P., Iannaccone, G., Papagiannaki, K., and Vinnakota, B. (2007). Towards an energy-star wlan infrastructure. In *Eighth IEEE Workshop on Mobile Computing Systems and Applications*.

[59] Jejurikar, R., Pereira, C., and Gupta, R. (2004). Leakage aware dynamic voltage scaling for real-time embedded systems. In *Design Automation Conference*.

[60] Jha, N. K. (2001). Low power system scheduling and synthesis. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*.

[61] Kandemir, M., Vijaykrishnan, N., Irwin, M. J., and Ye, W. (2000). Influence of compiler optimizations on system power. In *Proceedings of the 37th Annual Design Automation Conference*.

[62] Kim, J. and Rotem, D. (2011). Energy proportionality for disk storage using replication. In *Proceedings of the 14th International Conference on Extending Database Technology*.

[63] Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. (2007). kvm : the linux virtual machine monitor. In *Proceedings of the Linux Symposium*.

[64] Klausmeier, D. E. and Sathe, S. P. (1998). System for buffering data in the network having a linked list for each of said plurality of queues. US Patent 5,838,915.

[65] Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. *Oakland, CA : Analytics Press. August*.

[66] Koomey, J. G. (2007). Estimating total power consumption by servers in the u.s. and the world. Technical report, Lawrence Derkley National Laboratory.

[67] Koomey, J. G. (2008). Worldwide electricity used in data centers. *Environmental Research Letters*.

[68] Kouki, Y., Ledoux, T., Serrano, D., Bouchenak, S., Lejeune, J., Arantes, L., Sopena, J., Sens, P., et al. (2013). Sla et qualité de service pour le cloud computing. In *Conférence d'informatique en Parallélisme, Architecture et Système*.

[69] Kubo, R., Kani, J.-i., Ujikawa, H., Sakamoto, T., Fujimoto, Y., Yoshimoto, N., and Hadama, H. (2010). Study and demonstration of sleep and adaptive link rate control mechanisms for energy efficient 10g-epon. *Journal of Optical Communications and Networking*.

[70] Kultursay, E., Kandemir, M., Sivasubramaniam, A., and Mutlu, O. (2013). Evaluating stt-ram as an energy-efficient main memory alternative. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*.

[71] Kumar, S., Talwar, V., Kumar, V., Ranganathan, P., and Schwan, K. (2009). vmanage : loosely coupled platform and virtualization management in data centers. In *Proceedings of the 6th international conference on Autonomic computing*. ACM.

[72] Kwak, J. T. (2003). Low power type rambus dram.

[73] Leavitt, N. (2013). Hybrid clouds move to the forefront. *IEEE Symposium on Security and Privacy*.

[74] Lebeck, A. R., Fan, X., Zeng, H., and Ellis, C. (2000). Power aware page allocation. *ACM SIGPLAN Notices*.

[75] Leinenbach, D. and Santen, T. (2009). Verifying the microsoft hyper-v hypervisor with vcc. In *FM 2009 : Formal Methods*. Springer.

[76] Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. (2009). What's inside the cloud ? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*.

[77] Li, K. (2008). Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed. *Parallel and Distributed Systems, IEEE Transactions on*.

[78] Li, K., Kumpf, R., Horton, P., and Anderson, T. E. (1994). A quantitative analysis of disk drive power management in portable computers. In *USENIX Winter*.

[79] Li, X., Li, Z., David, F., Zhou, P., Zhou, Y., Adve, S., and Kumar, S. (2004). Performance directed energy management for main memory and disks. *ACM SIGARCH Computer Architecture News*.

[80] Li, X., Li, Z., Zhou, Y., and Adve, S. (2005). Performance directed energy management for main memory and disks. *Transaction on Storage*.

[81] Liu, J., Jaiyen, B., Veras, R., and Mutlu, O. (2012a). Raidr : Retention-aware intelligent dram refresh. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*.

[82] Liu, J., Zhao, F., Liu, X., and He, W. (2009). Challenges towards elastic power management in internet data centers. In *IEEE International Conference on Distributed Computing Systems Workshops*.

[83] Liu, S., Pattabiraman, K., Moscibroda, T., and Zorn, B. G. (2012b). Flikker : saving dram refresh-power through critical data partitioning. *ACM SIGPLAN Notices*.

[84] Liu, W., Wang, L., and Yin, J. (2011). Managing power consumption in a nic team.

[85] Lowe, S. (2011). *Mastering VMware vSphere 5*. Wiley. com.

[86] Majumdar, S. (2013). Management on clouds : the multifaceted problem and solutions.

[87] Malladi, K. T., Lee, B. C., Nothaft, F. A., Kozyrakis, C., Periyathambi, K., and Horowitz, M. (2012). Towards energy-proportional datacenter memory with mobile dram. In *Proceedings of the 39th International Symposium on Computer Architecture*.

[88] McGee, M., Chang, D., Madhi, N., and Reeves, M. (2009). Failover of multicast traffic flows using nic teaming.

[89] Meisner, D., Gold, B. T., and Wenisch, T. F. (2009). Powernap : eliminating server idle power. In *ACM Sigplan Notices*.

[90] Mell, P. and Grance, T. (2011). The nist definition of cloud computing (draft). *NIST special publication*.

[91] Mishchenko, D. (2010). *VMware ESXi : Planning, Implementation, and Security*. Course Technology Press.

[92] Mishra, R. and Jaiswal, A. (2012). Ant colony optimization : A solution of load balancing in cloud. *International Journal of Web & Semantic Technology (IJWesT)*.

[93] Miyakawa, D. and Ishikawa, Y. (2007). Process oriented power management. In *Industrial Embedded Systems, 2007. SIES'07. International Symposium on*.

[94] Mosberger, D. and Jin, T. (1998). httperf—a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*.

[95] Narayanan, D., Donnelly, A., and Rowstron, A. (2008). Write off-loading : Practical power management for enterprise storage. *ACM Transactions on Storage*.

[96] Nathuji, R., Isci, C., and Gorbatov, E. (2007). Exploiting platform heterogeneity for power efficient data centers. In *Fourth International Conference on Autonomic Computing*. IEEE.

[97] Nathuji, R. and Schwan, K. (2007). Virtualpower : coordinated power management in virtualized enterprise systems. *ACM SIGOPS Operating Systems Review*.

[98] Nathuji, R., Schwan, K., Somani, A., and Joshi, Y. (2009). Vpm tokens : virtual machine-aware power budgeting in datacenters. *Cluster computing*.

[99] Nedevschi, S., Popa, L., Iannaccone, G., Ratnasamy, S., and Wetherall, D. (2008). Reducing network energy consumption via sleeping and rate-adaptation. In *NSDI*.

[100] Nordman, B. and Christensen, K. (2005). Reducing the energy consumption of network devices. *Tutorial presented at the July*.

[101] Norris, C., Cohen, H., and Cohen, B. (2011). Leveraging ibm ex5 systems for breakthrough cost and density improvements in virtualized x86 environments. Technical report, White paper IBM.

[102] Oracle, V. (2011a). Virtualbox. *URL https ://www. virtualbox. org*.

[103] Oracle, V. (2011b). Virtualbox user manual.

[104] Osman, S., Subhraveti, D., Su, G., and Nieh, J. (2002). The design and implementation of zap : a system for migrating computing environments. In *Proceedings of the 5th symposium on Operating systems design and implementation*. ACM.

[105] Pabla, C. S. (2009). Completely fair scheduler. *Linux Journal*.

[106] Pallipadi, V. and Starikovskiy, A. (2006). The ondemand governor. In *Proceedings of the Linux Symposium*.

[107] Pandey, V., Jiang, W., Zhou, Y., and Bianchini, R. (2006). Dma-aware memory energy management. In *HPCA*.

[108] Park, D. J., Kim, K., and Lee, P. J. (2005). Public key encryption with conjunctive field keyword search. In *Information Security Applications*.

[109] Patel, R. and Patel, S. (2013). Survey on resource allocation strategies in cloud computing. *International Journal of Engineering*.

[110] Patil, R., Reddy, M., Biradar, V., et al. (2013). Optimization of energy consumption in ethernet using new switch architecture. *World Journal of Science and Technology*.

[111] Patterson, D. A., Gibson, G., and HKatz, R. (1988). *A case for redundant arrays of inexpensive disks (RAID)*.

[112] Popek, G. J. and Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. In *Commun. ACM*.

[113] Raghunathan, A., Segev, G., and Vadhan, S. (2013). Deterministic public-key encryption for adaptively chosen plaintext distributions. In *Advances in Cryptology–EUROCRYPT 2013*. Springer.

[114] Raj, H., Nathuji, R., Singh, A., and England, P. (2009). Resource management for isolation enhanced cloud services. In *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM.

[115] Rao, N. V. and MeeraSaheb, S. (2013). A survey of cloud computing : Cloud computing concerns and issues. *International Journal of Engineering*.

[116] Rutkowska, J. (2006). Subverting vistatm kernel for fun and profit. *Black Hat Briefings*.

[Salesforce] Salesforce. Salesforce.com.

[118] Schulz, G. (2008). Maid 2.0 : Energy savings without performance compromises. *Greg Schulz, Jan*.

[119] Services, M. (2007). Server consolidation using virtualization. Technical report, Microsoft Services.

[120] Shah, A. (2008). Kernel-based virtualization with kvm. *Linux Magazine*.

[121] Sharma, P., Banerjee, S., and Ranganathan, P. (2013). Heterogeneous network switch system.

[122] Shin, D. W. (2004). Rambus dram.

[123] Shuja, J., Madani, S. A., Bilal, K., Hayat, K., Khan, S. U., and Sarwar, S. (2012). Energy-efficient data centers. *Computing*.

[124] Sprint, I. (2007). The applied research group.

[125] Srikantaiah, S., Kansal, A., and Zhao, F. (2008). Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*.

[126] Stanoevska-Slabeva, K., Wozniak, T., and Ristol, S. (2009). *Grid and cloud computing : a business perspective on technology and applications*.

[127] Stoess, J., Lang, C., and Bellosa, F. (2007). Energy management for hypervisor-based virtual machines. In *USENIX annual technical conference*.

[128] Sudan, K. (2013). *DATA PLACEMENT FOR EFFICIENT MAIN MEMORY ACCESS*. PhD thesis, The University of Utah.

[129] Sugerman, J., Venkitachalam, G., and Lim, B.-H. (2001). Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *USENIX Annual Technical Conference, General Track*.

[130] Suleiman, D., Ibrahim, M., and Hamarash, I. (2005). Dynamic voltage frequency scaling (dvfs) for microprocessors power and energy reduction. In *4th International Conference on Electrical and Electronics Engineering*.

[131] Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C., Anderson, A. V., Bennett, S. M., Kagi, A., Leung, F. H., and Smith, L. (2005). Intel virtualization technology. *Computer*.

[132] Uhlig, V., LeVasseur, J., Skoglund, E., and Dannowski, U. (2004). Towards scalable multiprocessor virtual machines. In *Virtual Machine Research and Technology Symposium*.

[133] Valentini, G. L., Lassonde, W., Khan, S. U., Min-Allah, N., Madani, S. A., Li, J., Zhang, L., Wang, L., Ghani, N., Kolodziej, J., et al. (2013). An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*.

[134] Van, H. N., Tran, F., and Menaud, J.-M. (2009). Sla-aware virtual resource management for cloud infrastructures. In *IEEE International Conference on Computer and Information Technology*.

[135] van Doorn, L. (2006). Hardware virtualization trends. In *VEE*, volume 6.

[136] van Surksum, K. (2013). Paper : The cpu scheduler in vmware vsphere 5.1.

[137] Velte, A. and Velte, T. (2009). *Microsoft virtualization with Hyper-V*. McGraw-Hill, Inc.

[138] Verma, A., Koller, R., Useche, L., and Rangaswami, R. (2010). Srcmap : energy proportional storage using dynamic consolidation. In *Proceedings of the 8th USENIX conference on File and storage technologies*.

[139] VirtualBox, I. (2008). The virtualbox architecture.

[140] Virtualization, A. (2005). Secure virtual machine architecture reference manual. *AMD Publication*.

[141] VMWare, I. and ESXi, V. (2009). Vmware esx server.

[142] von Laszewski, G., Wang, L., Younge, A. J., and He, X. (2009). Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*.

[143] Voorsluys, W., Broberg, J., and Buyya, R. (2011). Introduction to cloud computing. *Cloud computing : Principles and paradigms*.

[144] Waldspurger, C. A. and Weihl, W. E. (1994). Lottery scheduling : Flexible proportional-share resource management. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*.

[145] Waldspurger, C. A. and Weihl, W. E. (1995). Stride scheduling : Deterministic proportional-share resource management. Technical report, Technical Memo MIT/LCS/TM-528, MIT Laboratory for Computer Science.

[146] Ward, B. (2002). *The book of VMware : the complete guide to VMware workstation*. No Starch Press.

[147] Watson, J. (2008). Virtualbox : bits and bytes masquerading as machines. *Linux Journal*.

[148] Weissel, A., Beutel, B., and Bellosa, F. (2002). Cooperative i/o : A novel i/o semantics for energy-aware applications. In *Proceedings of the 5th symposium on Operating systems design and implementation*. ACM.

[149] Wen, C., He, J., Zhang, J., and Long, X. (2010). Pcfs : Power credit based fair scheduler under dvfs for muliticore virtualization platform. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*. IEEE.

[150] Wu, D., He, B., Tang, X., Xu, J., and Guo, M. (2012). Ramzzz : rank-aware dram power management with dynamic migrations and demotions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*.

[151] Yu, Y., Ni, J., Yang, H., Mu, Y., and Susilo, W. (2013). Efficient public key encryption with revocable keyword search. *Security and Communication Networks*.

[152] Zhang, X. and Dong, Y. (2008). Optimizing xen vmm based on intel® virtualization technology. In *Internet Computing in Science and Engineering, 2008. ICICSE'08. International Conference on*. IEEE.

[153] Zhang, X., Du, H.-t., Chen, J.-q., Lin, Y., and Zeng, L.-j. (2011). Ensure data security in cloud storage. In *Proceedings of the 2011 International Conference on Network Computing and Information Security - Volume 01*.

[154] Zhao, B., Aydin, H., and Zhu, D. (2013). Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints. *ACM Trans. Des. Autom. Electron. Syst.*

[155] Zhou, P., Pandey, V., Sundaresan, J., Raghuraman, A., Zhou, Y., and Kumar, S. (2004). Dynamic tracking of page miss ratio curve for memory management. In *ACM SIGOPS Operating Systems Review*.

[156] Zhu, Q. and Zhou, Y. (2005). Power-aware storage cache management. *Computers, IEEE Transactions on*.

Liste des tableaux

Table des figures