THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :
Damien Borgetto
le 03 Juin 2013

Titre :
Allocation et Réallocation de Services pour les Économies d'Énergie dans les Clusters et les Clouds

---

École doctorale et discipline ou spécialité :
ED MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de recherche :
IRIT - Institut de Recherche en Informatique de Toulouse

Directeur(s) de Thèse :
Jean-Marc PIERSON
Georges DA COSTA

Jury :
Mr Pascal BOUVRY - Rapporteur
Mme Christine MORIN - Rapporteur
Mr Jean-Marc MENAUD - Examinateur
Mr Olivier BEAUMONT - Examinateur
Mr Jean-Marc PIERSON - Directeur de thèse
Mr Georges DA COSTA - CoDirecteur de thèse

Table des matières

Résumé

L'informatique dans les nuages (cloud computing) est devenu durant les dernières années un paradigme dominant dans le paysage informatique. Son principe est de fournir des services décentralisés à la demande, et permet aux clients de payer uniquement pour ce dont ils ont besoin. La demande croissante pour ce type de service amène les fournisseurs de service clouds à augmenter la taille de leurs infrastructures à tel point que les consommations d'énergie ainsi que les coûts associés deviennent très importants.

Chaque fournisseur de service cloud doit répondre à des demandes différentes. Les gestionnaires d'infrastructure doivent héberger un ensemble de ces services. C'est pourquoi au cours de cette thèse, nous nous sommes intéressés à la gestion des ressources efficace en énergie dans les clouds.

Pour ce faire, nous avons tout d'abord modélisé et étudié le problème de l'allocation de ressources initiale en fonction des services, en calculant des solutions approchées via des heuristiques, puis en les comparant à la valeur optimale calculée grace à la résolution d'un programme linéaire associé.

Nous avons ensuite étendu notre modèle de ressources pour nous permettre d'avoir une solution plus globale, en y intégrant de l'hétérogénéité entre les machines et des infrastructures de refroidissement. Nous avons enfin validé notre modèle par simulation.

Les services doivent habituellement faire face à différentes phases de charge, ainsi qu'à des pics d'utilisation. C'est pourquoi, nous avons étendu le modèle d'allocation de ressources pour y intégrer la dynamicité des requêtes et de l'utilisation des ressources, ainsi que la notion de coût d'allumage, d'extinction des serveurs et le coût de migration des services d'un serveur vers un autre. Nous avons mis en œuvre une infrastructure de cloud simulée, visant à contrôler l'exécution des différents services ainsi que le placement de ceux-ci. Ainsi notre approche permet de réduire la consommation d'énergie globale de l'infrastructure, ainsi que de limiter autant que possible les dégradations de performance.

Abstract

Cloud computing has become over the last years an important paradigm in the computing landscape. Its principle is to provide decentralized services and allows client to consume resources on a pay-as-you-go model. The increasing need for this type of service brings the service providers to increase the size of their infrastructures, to the extent that energy consumptions as well as operating costs are becoming important.

Each cloud service provider has to provide for different types of requests. Infrastructure manager then have to host all the types of services together. That's why during this thesis, we tackled energy efficient resource management in the clouds.

In order to do so, we first modeled and studied the initial service allocation problem, by computing approximated solutions given by heuristics, then comparing it to the optimal solution computed with a linear program solver.

We then extended the model of resources to allow us to have a more global approach, by integrating the inherent heterogeneity of clusters and the cooling infrastructures. We then validated our model via simulation.

Usually, the services must face different stages of workload, as well as utilization spikes. That's why we extended the resource allocation model to include dynamicity of requests and resource usage, as well as the concept of powering on or off servers, or the cost of migrating a service from one host to another. We implemented a simulated cloud infrastructure, aiming at controlling the execution of the services as well as their placement. Thus, our approach enables the reduction of the global energy consumption of the infrastructure, and limits as much as possible degrading the performances.

Remerciements

Je tiens tout d'abord à remercier Jean-Marc Pierson et Georges Da Costa, mes directeurs de thèse, qui m'ont guidé et encouragé depuis mon premier stage jusqu'à la fin de ma thèse, et m'ont permis de mener mon travail jusqu'au bout. Je tiens à les remercier tout particulièrement pour la confiance qu'ils m'ont accordés, et le temps accordé malgré un emploi du temps chargé.

Un grand merci à Pascal Bouvry, qui m'a aussi fait l'honneur d'accepter d'être rapporteur de cette thèse. Merci aussi de m'avoir accueilli chaleureusement pendant une semaine dans votre équipe dans le cadre de l'action COST IC0804.

Un tout aussi grand merci à Christine Morin, qui m'a fait l'honneur d'accepter d'être rapporteur pour cette thèse, pour ses remarques justes et pour avoir accepté de charger un peu plus son calendrier de déplacement en France.

Je tiens à remercier Olivier Beaumont et Jean-Marc Menaud, qui ont accepté d'être examinateur pour la soutenance.

Un grand merci à toutes les personnes que j'ai côtoyé au cours de ces quatre dernières années, et avec qui j'ai eu la chance de travailler : Patricia, François, Saif, Leandro, Thiam, Christina, Tom, Cédric, Nebyou. Sans oublier Frédéric avec qui nous avons eu plusieurs discussions très intéressantes lors de ma visite au Luxembourg et Michael, que j'estime particulièrement pour la période passée à travailler ensemble pendant et après sa visite à Toulouse.

Merci à ma famille, ma mère Anne et mon père Christian qui a subi la fastidieuse tâche de relire mon manuscrit, ma soeur Florence qui a pavé la voie devant ma thèse en en faisant une en premier, mon frère Emmanuel et mon beau-frère Jonathan qui m'ont soutenu et encouragé pendant les bons comme les mauvais moments. Je remercie aussi Alexia et sa famille, Benoit, Françoise et Cyril, qui ont aussi toujours été là pendant ces 4 dernières années.

Merci à mes amis, qui ont malgré eux accepté de ne plus trop me voir pendant la dernière année, mais qui m'ont quand même soutenu : Pierro, Yann, Julien, Guido, Marjolaine, Albin, Gaspard, Rémi, Bénédicte, Max, Guillaume, Mathilde, et tous ceux que j'ai oubliés mais que je n'estime pas moins.

Merci enfin à tous ceux qui viendront à ma soutenance, même si j'aimerais secrètement qu'il n'y ait personne, ça me touche beaucoup.

Enfin, j'aimerais dédier cette thèse à mon autre moitié (voire deux tiers), Alexia, qui m'a supporté et me supporte encore, et m'a soutenu durant ces dernières années, et m'a confectionné des gâteaux pour accompagner mes soirées de travail.

Un grand merci à tous, du fond du cœur.

Chapitre 1

Introduction

1.1 Contexte et Motivation

L'informatique dans les nuages (appelé cloud computing en anglais et dans la suite du manuscrit) s'est imposé ces dernières années comme un paradigme majeur d'utilisation des infrastructures informatiques. Celui-ci répond à des besoins et demandes croissantes en terme de disponibilité, scalabilité et flexibilité. Les fournisseurs de services de clouds permettent à leur client de payer directement en fonction de ce qu'ils souhaitent utiliser.

Maintes définitions de ce qu'est un cloud ont été données au cours des années, et nous retiendrons celle donnée par Vaquero et al. dans [102], qui traduite de l'anglais est comme suit :

"Les clouds sont un large ensemble de ressources virtualisées facilement utilisables et accessibles (telles que materielles, plateformes de développement et/ou de services). Ces ressources peuvent être dynamiquement reconfigurées pour s'adapter à une charge (ou une échelle) variable, permettant une utilisation des ressources optimum. Cet ensemble de ressources est en général exploité via un modèle de paiement à l'usage dans lequel le fournisseur de l'infrastructure donne des garanties au moyen de contrats de service (appelés Service Level Agreement ou SLA)."

Il existe plusieurs types de clouds qui fournissent un niveau différent de service à l'utilisateur. En premier, les clouds de type *Software as a Service (SaaS)* qui fournissent directement un accès à distance à des applications. Des services tels que Gmail [1] qui permet un accès à distance à une application de gestion d'e-mail, et Surveymonkey [2] qui permet un accès à une application de distribution et analyse de sondages directement sur internet sont considérés comme des clouds SaaS.

Viennent ensuite les types clouds permettant à l'utilisateur d'avoir accès à des outils pour coder et déployer des applications de manière rapide et efficace. Ceux-ci sont appelés *Platform as a Service (PaaS)*. L'AppEngine [3] de Google, Windows Azure [4] de Microsoft et OpenShift de RedHat en sont des exemples [5].

Enfin, le type de clouds qui va majoritairement nous intéresser, est celui des *Infrastructure as a Service (IaaS)*, qui fournissent l'accès à une infrastructure. Cela se traduira par la location de serveurs (virtuels ou non) ainsi que les infrastructures sous-jacente comme le stockage ou le réseau. Rackspace Cloud [6] est un exemple de cloud de type IaaS, ou encore un des plus gros

1. GMail - https ://mail.google.com/
2. SurveyMonkey - htto ://www.surveymonkey.com
3. AppEngine - https ://developers.google.com/appengine/
4. Windows Azure - www.windowsazure.com/
5. OpenShift - https ://openshift.redhat.com/app/
6. RackSpace - http ://www.rackspace.com

acteurs du secteur Amazon Elastic Cloud Computing (EC2) [7] fournissent de tels services. Il est intéressant de noter que la plateforme grid5000 [3] propose des services similaire à une infrastructure de cloud IaaS, dans la mesure où l'on peut réserver un certain nombre de machines et les utiliser pour nos expériences en tant que plateforme d'expérimentations. Cependant, grid5000 ne tombera pas dans la classification de cloud IaaS puisqu'elle ne respectera pas directement les considérations de dimensionnement et d'adaptabilité. Grid5000 restera par contre historiquement une des premières infrastructures permettant la réservation de machines pour une utilisation en tant qu'infrastructure.

Tout ces services ont pris une importance grandissante ces dernières années, et ont vu leurs demandes augmenter de facon significative à tel point que pour faire face à l'explosion de la demande les fournisseurs de services doivent se doter de nombreuses machines pour avoir suffisamment de puissance de calcul. Ainsi, les infrastructures deviennent telles que la consommation électrique requise pour faire fonctionner ces parcs de serveurs devient une part importante du coût total de fonctionnement de ces serveurs. C'est pourquoi, les fournisseurs de clouds s'intéressent aux outils et stratégies permettant de réduire leur facture d'électricité et de minimiser les coûts.

1.1.1 Consommation des Centres de Données

La consommation électrique des centres de données (*data centers* en anglais) dans le monde a été estimée pour 2010 entre 1.1% et 1.5% de la consommation électrique globale. Même si cela parait peu en proportion, cette consommation d'énergie représente plus de 200 milliards de kilowatt-heure[67]. Les grappes de serveurs (appelées clusters en anglais et par la suite) de nos jours contiennent un nombre de serveurs de l'ordre de la dizaine pour les plus petits, et de plusieurs centaines de milliers pour les plus gros [66].

Cette consommation d'énergie est en croissance permanente au fil des années. Elle était estimée à 12% par an en 2007 par l'U.S Environmental Protection Agency pour atteindre un coût de l'électricité utilisée par les serveurs des data centers estimé de 7.4 milliards de dollars [8] et l'équivalent de 16 fois la production du réacteur nucléaire *CHOOZ-B-1* de Charleville en France [7] en 2012.

Si l'on regarde par contre la consommation individuelle des serveurs, nous avons une consommation qui sera en moyenne aux alentours de 50W lorsque la machine n'effectue aucun calcul, et 250W lorsqu'elle est chargée à 100%[34]. Nous pouvons donc avoir des data centers consommant une puissance instantanée de plusieurs dizaines de mégawatts. Et ceci, sans compter toute l'infrastructure nécessaire au fonctionnement des serveurs, comme l'infrastructure de refroidissement par exemple.

La consommation globale en énergie d'une entreprise possédant beaucoup de serveurs peut donc devenir très importante. L'exemple d'un des géants d'internet Google en est la preuve. Google possédait déjà en 2009 plus de 900 000 serveurs, et a rapporté une consommation de 2 675 898 MWh en 2011 [52], ce qui représente, sachant que Google a réduit son empreinte carbone de moitié entre 2007 et 2011, 1.68 millions de tonnes métrique de CO2. À titre de comparaison, la centrale nucléaire française de Belleville a produit 18 267 160 MWh pendant l'année 2011[7].

Comme mentionné précédemment, les serveurs de calculs ne sont pas les seuls à consommer de l'énergie dans les clusters. En effet, pour calculer la consommation énergétique d'un data center, il faut aussi prendre en compte le refroidissement, l'équipement réseau, l'alimentation électrique de l'infrastructure. Des études ont été réalisées sur la répartition de la consommation d'électricité dans les clusters comme montré dans [86, 84].

7. Amazon EC2 - http ://aws.amazon.com/fr/ec2/

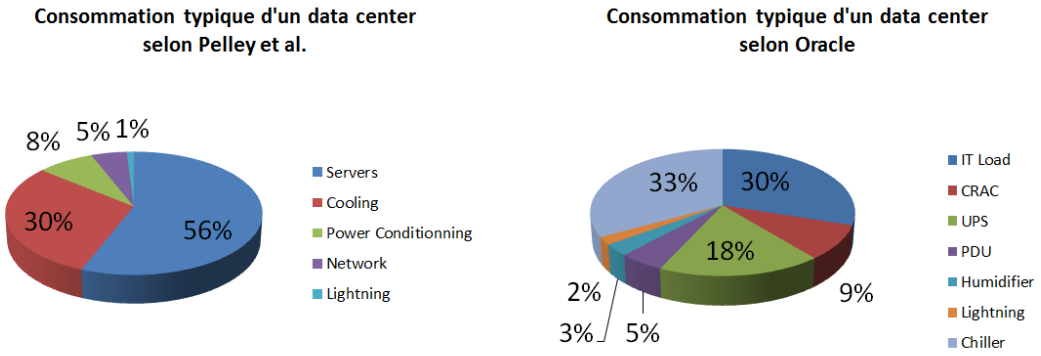


FIGURE 1.1 – Différentes distributions de l'électricité dans les data centers : Pelley et al.[86] à gauche et Oracle [84] à droite

Bien que les chiffres varient en fonction des différents data centers, certaines tendances peuvent être remarquées. Comme on peut le voir dans la figure 1.1, les deux portions majoritairement consommatrices d'électricité sont les serveurs et le refroidissement. Il est bon de noter que ces deux parties des data centers peuvent être gérées au niveau du système de gestion. Non que cela ne soit pas possible pour les autres, mais même si on peut réduire l'énergie consommée par l'unité d'approvisionnement d'électricité en changeant le matériel ou la manière dont il est utilisé, les actions sur les serveurs par exemple sont plus larges et peuvent être faites de manière automatique par une application, qui va changer la fréquence d'opération du matériel, voire même éteindre et allumer les machines. Ainsi, le potentiel d'économies d'énergie au niveau des machines et du refroidissement est plus grand d'un point de vue applicatif, du fait de leur part importante dans la distribution de la consommation des data centers.

1.1.2 Pourquoi réduire la consommation

La consommation électrique des data centers n'a cessé de croître ces dernières années, en suivant la courbe de progression de la performance de calcul [68]. Cependant, de la même manière que pour les processeurs, nous allons heurter le mur de la consommation. Lorsque le temps était à la course à la miniaturisation il est apparu que la densité de chaleur générée par le composant imposait une contrainte forte sur la densité des transistors. Un parallèle peut être fait avec les clusters et la consommation d'électricité qu'ils génèrent.

Ainsi, les clusters ont augmenté leur nombre de serveurs en même temps que leur puissance. Ce qui nous amène à faire des projections pour le futur de la consommation des data centers, qui sont du même ordre que le mur de la miniaturisation des composants. En effet, la miniaturisation croissante des processeurs a dû ralentir du fait de la dissipation de chaleur insuffisante, et a amené les constructeurs à repenser leurs architectures en augmentant le nombre de processeurs ou de coeurs. Si on projette la consommation électrique des superordinateurs du top500[98] en supposant qu'elle croît de façon constante, on arrivera à des clusters d'une consommation de plusieurs centaines de mégawatts ! Ce qui réprésente la nécessité d'avoir un réacteur nucléaire pour alimenter un seul des supercalculateurs.

Il est donc nécessaire de réduire cette consommation par des moyens permettant à la fois de réduire l'électricité demandée, mais aussi de garder un niveau acceptable de performance, car bien qu'éteindre systématiquement la moitié des serveurs permette de réduire la consommation

d'énergie, ça n'est évidemment pas une solution viable. Ajoutons à cela l'impact environnemental des data center. Bien que décentraliser des opérations qui étaient faites localement permette de réduire la consommation d'énergie et l'empreinte carbone en délocalisant des applications vers des serveurs globalement plus efficaces énergétiquement parlant, il n'en reste pas moins que la consommation globale des data centers étaient responsables en 2007 de 116 tonnes métriques d'équivalent dioxide de carbone (MtCO2) selon un rapport de Greenpeace [62]. Ce même rapport estime que ces mêmes émissions pourraient doubler d'ici à 2020 pour atteindre 257 MtCO2.

La consommation d'électricité des serveurs n'est en effet pas la seule source d'émission CO2, il faut aussi prendre en compte le cycle de vie complet d'un serveur, depuis le moment où le serveur est construit jusqu'à son démantèlement, un serveur sera responsable de plus d'émission que sa seule consommation d'électricité.

Enfin, et c'est peut-être ce qui intéresse le plus les possesseurs de clouds, l'impact sur le coût total des infrastructures, autrement dit le coût en électricité de l'exploitation d'un data center, est important. Comme mentionné précédemment, ce coût peut rapidement devenir important, jusqu'à ce qu'alimenter un data center en électricité sur plusieurs années revienne au même prix qu'acheter ce même data center.

Que ce soit donc pour des raisons économiques, environnementales ou tout simplement pratiques, il est intéressant, voire dans certains cas nécessaire, de réduire la consommation énergétique des clouds. Cependant, comme nous allons le voir par la suite, cela ne peut pas se faire de n'importe quelle façon.

1.2 Problématique

La problématique est donc la suivante : comment réduire la consommation d'énergie des serveurs sans pour autant trop impacter la qualité de service et les performances des applications qui s'exécutent sur le cloud ? Il existe de nombreux moyens de réduire la facture d'électricité, et bon nombre peuvent se faire sans impacter la qualité de service. Cependant, en acceptant une dégradation faible de performance, il est possible de réduire encore plus la puissance consommée par l'infrastructure.

1.2.1 Comment réduire la consommation

La première question qu'il faut donc se poser est comment réduire la consommation d'énergie. Différents moyens sont à notre disposition, certains étant au niveau du fonctionnement de la machine elle même, et donc à une granularité de phases d'application, alors que d'autres sont à des niveaux plus hauts, et donc à une granularité d'application entière. Certains sont basés sur des leviers matériels, alors que d'autres sont purement applicatifs.

Action sur le processeur

Une des méthodes les plus répandues pour controller la consommation d'une application est celle du Dynamic Voltage and Frequency Scaling (DVFS), qui consiste simplement à changer le couple fréquence/voltage d'un processeur, afin de le faire tourner moins vite lorsque l'application qui s'exécute a besoin de moins de CPU. Habituellement, ce phénomène intervient par exemple lorsqu'une application a fini de calculer et envoie ou reçoit des données. Dans ce cas, nous pouvons facilement définir cette phase comme étant capable de s'exécuter à une fréquence plus faible. Cependant, il n'est pas toujours aussi direct de choisir comment et quand réduire la fréquence du processeur, car cela nécessite que les applications soient connues, et cela nécessite aussi d'éviter de faire des mauvais changements. En effet, une mauvaise gestion du DVFS peut amener à ne

pas économiser d'énergie, voire perdre de l'efficacité énergétique lorsque l'augmentation de temps pris par une tâche n'est pas compensé par une baisse suffisante en consommation de puissance.

Action sur les machines

Des actions sont aussi possibles au niveau des machines. En effet, la manière la plus simple de réduire la consommation des serveurs est simplement de les éteindre. Cela n'est pourtant pas aussi simple et direct en pratique. Ainsi, pour pouvoir éteindre un serveur, il faut d'abord s'assurer qu'il n'est pas utilisé sur le moment, mais aussi que l'on ne vas pas en avoir besoin dans un futur immédiat, sous peine de se retrouver avec un manque de ressources, et ainsi de ne plus être capable de satisfaire les demandes des clients. On pourra cependant passer les machines dans plusieurs états, allant d'allumé à éteint en passant par hibernation ou *suspend-to-RAM*[39, 29]. Ces différents états discrets d'une machine correspondent en fait à un différent sous-ensemble de composants qui vont rester allumés, afin de répondre plus vite aux nouvelles requêtes des clients. La composante temporelle joue ici un rôle crucial, puisque chaque action aura une longueur en temps associée, pendant laquelle elle induira un surcoût et ne sera pas utilisable pour satisfaire des demandes de clients.

Les actions sur les machines restent cependant des leviers très importants pour économiser de l'énergie, puisque la majorité de la consommation d'un serveur vient simplement du fait d'être allumé. Autrement dit, la portion de consommation correspondant au surcoût induit par l'exécution d'applications sur une machine est inférieure à la portion de consommation induite par l'état allumé de cette même machine[34].

Action sur l'infrastructure

Les actions sur les machines ont aussi une portée plus grande, car si l'on se place au niveau d'un cluster, plus le nombre de serveurs allumés est réduit, plus l'infrastructure apparentée peut économiser de l'énergie. L'exemple le plus probant est celui de l'infrastructure de refroidissement d'un data center.

Dans un data center, l'infrastructure de refroidissement composée de CRAC (Computer Room Air Conditionner, qui sert à rafraichir l'air utilisé pour refroidir les serveurs) et de CRAH (Computer Room Air Handler, qui achemine l'air jusqu'aux serveurs) tente de maintenir la température dans la salle de serveurs à un certain niveau, et si possible de manière homogène. Si l'on veut éviter les points chauds (ou *hotspots*), il faudra augmenter la puissance de la ventilation afin d'homogénéiser la température autant que possible dans la pièce. De la même manière, si la température moyenne de la pièce augmente, il faudra augmenter la puissance de l'infrastructure de refroidissement afin de refaire descendre la température au palier souhaité.

Nous voyons bien ici que la température se voit intimement liée à la charge des serveurs. Si le data center se voit attribué un grand nombre de calculs à effectuer, les serveurs vont tourner à plein régime, rejettant ainsi de la chaleur dûe à leur consommation d'énergie augmentée, ce qui va faire monter la température de la pièce et/ou induire des points chauds, l'infrastructure de refroidissement va tenter de compenser en augmentant sa propre puissance, consommant ainsi encore plus d'énergie.

Le comportement de l'infrastructure de refroidissement est donc intimement liée à la charge des serveurs. Il est par conséquent possible d'utiliser ce fait pour induire des économies d'énergie par une gestion intelligente de la charge de travail pour éviter d'avoir des points chauds, ou encore positionner la charge de manière à pouvoir éteindre une partie des CRAC/CRAH.

Action sur les applications

Enfin, il est possible d'agir sur la consommation d'énergie au niveau applicatif. Lorsqu'un gestionnaire de cloud met en place son infrastructure, elle est faite pour répondre à des demandes d'exécutions d'applications. Ce faisant, le fournisseur de services doit, en respectant les demandes des clients, faire le lien entre son infrastructure, et les tâches à exécuter. Autrement dit, il doit faire l'allocation de ces tâches sur les différents serveurs de l'infrastructure en prenant en compte les métriques qui lui sont importantes, que ce soit le coût, l'énergie, ou la performance par exemple.

Cette allocation est un processus décisionnel purement applicatif qui va viser à décider où exécuter des tâches, mais aussi quand les exécuter et comment. Une distinction est à faire entre ces trois aspects. En effet, décider de l'endroit de l'exécution d'une tâche va permettre de ne pas surcharger certaines machines pour éviter des points chauds, ou allouer en priorité sur les machines qui sont le plus énergétiquement efficaces. Décider de quand exécuter une tâche par contre va permettre de ne pas forcément exécuter les tâches dans l'ordre d'arrivée, mais plutôt de tenter d'optimiser le temps d'exécution global de ces tâches. Chacune aura potentiellement une durée différente, et réorganiser l'ordre des tâches permettra un temps d'exécution total des tâches plus court, économisant ainsi de l'énergie. Enfin, la décision de comment exécuter les tâches nous ramène par exemple aux techniques de DVFS, où l'on va décider de la fréquence à laquelle doit se trouver un serveur.

1.2.2 Problèmes associés

De toutes les actions que nous pouvons entreprendre, toutes sont à utiliser avec parcimonie car elles ne sont pas sans comporter des désavantages. Ainsi, une mauvaise utilisation des leviers de réduction de la consommation d'énergie entrainera un effet contre productif qui peut se révéler très problématique. L'exemple le plus simple serait d'éteindre et de rallumer une machine trop souvent, faisant de la machine un serveur allumé en permanence mais peu ou pas utilisable, et par conséquent extrêmement inéfficace en terme de performance et d'énergie.

Chacun des leviers possède donc ses propres avantages et défauts, faisant du problème de la gestion des tâches dans un cloud un processus compliqué. Prenons par exemple le cas du DVFS, qui permet de réduire grandement la consommation du processeur. L'application d'un changement de fréquence n'est pas instantanée, et passer d'une fréquence à une autre prend un peu de temps. Par conséquent, de la même manière que pour les machines, des changements de fréquence trop rapproché ralentira fortement l'exécution de l'application, entrainant ainsi une surconsommation. De plus, les changements trop fréquents ne sont pas le seul problème lié au DVFS, car encore faut-il utiliser la bonne fréquence. Une mauvaise fréquence d'allocation fera consommer plus au programme, et les plus petites fréquences ne seront pas forcément les plus efficaces en énergie. Si une application s'exécute en un certain temps à une fréquence donnée, baisser la fréquence d'exécution peut rallonger le temps d'exécution de l'application à un point tel que l'énergie consommée sera au final plus grande.

Enfin, une bonne utilisation du DVFS reposera sur une bonne connaissance de l'application que l'on veut gérer. Il est en effet possible d'effectuer des actions en fonction des tendances mesurées des applications, si l'on connait la tâche à exécuter. Nous aurons ainsi de meilleurs résultats au final, puisque l'on sera capable d'affecter à tout instant la bonne fréquence, et d'anticiper les changements de charge trop rapides.

Cette connaissance s'applique aussi aux autres leviers, puisqu'une bonne utilisation des capacités de refroidissement d'un cluster reposera sur la bonne connaissance de la localisation spatiale de l'infrastructure, afin de pouvoir prévenir les points chauds à certains endroits, ou encore choisir des CRAC/CRAH à ne pas modifier pour éviter de trop perturber les flux d'air bien spécifique

du data center. Un autre exemple est qu'au niveau de l'allocation temporelle, dans le cas où l'on cherche à choisir quand allouer les tâches, une bonne connaissance de la charge de travail nous permet d'identifier des phases dans une journée, afin de pouvoir au mieux répondre à la demande.

Lors de la gestion d'un data center, il est donc intéressant d'utiliser tous les moyens qui sont à notre disposition, en prenant bien en compte que bien que ces moyens nous permettent des économies d'énergie, il faut les utiliser avec une bonne connaissance des infrastructures et des effets sous-jacents à chacun. Chacune des techniques ayant un potentiel pour améliorer l'efficacité énergétique d'un data center, ainsi qu'ayant potentiellement un effet négatif sur la performance des tâches.

1.3 De l'allocation statique à la réallocation

Dans cette thèse, nous proposons de réduire la consommation d'énergie en gardant un niveau acceptable de qualité de service via la gestion de l'allocation des tâches sur les différents serveurs. Cela nous permettra de réduire la consommation d'énergie en utilisant les leviers d'allumage et d'extinction des hôtes, mais aussi de l'allocation des tâches aux endroits les plus efficaces énergétiquement parlant.

Afin d'allouer les tâches de manière cohérente, nous avons commencé par modéliser le problème en tant qu'ensemble de contraintes linéaires, modèle qui servira de base pour l'élaboration d'algorithmes, que des bornes, solutions optimales et approchées, pour la comparaison des performances de ces mêmes algorithmes.

Le problème de l'allocation efficace en énergie présente le problème d'être une optimisation multi objectifs, qui doit équilibrer la performance des applications d'une part, et la consommation globale de l'infrastructure d'autre part. Le problème étant que les deux objectifs sont très souvent antagonistes, et améliorer l'un fera baisser l'autre. Pour tenter de palier à ce problème, nous proposons de décomposer le problème de l'allocation de tâches en trois problèmes distincts.

Le premier sera le problème borné en énergie ou en puissance électrique, et correspondra aux cas où l'on veut éviter que l'infrastructure dépasse une certaine demande en électricité, pour des raisons de coûts ou d'infrastructure. Pour ce problème là, nous supposerons que le fournisseur de services sait définir la valeur qu'il ne veut pas dépasser, ou que cette valeur est calculée par quelqu'un d'autre. Nos algorithmes doivent alors calculer une allocation proche de l'optimal en respectant la performance des différentes tâches, tout en respectant la contrainte d'énergie.

Le second problème que nous proposons représente le cas où nous voulons respecter un certain niveau de performance pour tous les tâches, autrement dit avoir au minimum une certaine satisfaction de ce qui a été alloué par rapport à ce qui avait été demandé par les tâches. Sous cette contrainte, les différents algorithmes vont tenter de trouver une allocation permettant d'optimiser la consommation d'énergie de l'infrastructure.

Le dernier problème est celui du multi objectifs. Pour celui-ci, nous considérons que l'on veut économiser de l'énergie, sans avoir de contrainte dure sur celle-ci, mais nous voulons le faire sans trop atteindre au bon fonctionnement des applications, puisqu'une mauvaise exécution d'une application aura un impact négatif pour le fournisseur de services[46]. Pour résoudre ce problème, nous allons optimiser une combinaison linéaire de la performance et de l'énergie.

Une fois les différents problèmes définis, nous avons étendu notre modèle à deux aspects. Un des aspects est lié au fait qu'un cluster appartenant à un cloud n'est que rarement complètement homogène. Il arrive ainsi souvent que l'on doive remplacer des parties de l'infrastructure pour maintenance, ou en rajouter pour augmenter la capacité. Cette hétérogénéité nous demande de connaitre notre infrastructure, pour allouer les tâches sur des serveurs qui n'ont pas forcément

la même capacité, en plus de la même localisation, et par conséquent pas les mêmes efficacités énergétiques.

L'autre aspect est celui du refroidissement. Comme vu précédemment dans la figure 1.1, le refroidissement prend une part conséquente dans la consommation d'un data center. Si celle-ci peut paraitre indépendante de l'allocation de tâches, elle n'en reste pas moins liée. Prendre en compte l'infrastructure de refroidissement nous permet, avec une bonne connaissance des data centers, d'être capable d'agir dessus en fonction de la charge courante des serveurs gérés par un certain CRAC/CRAH.

Nous avons donc intégré ces deux aspects dans notre modèle de contraintes ainsi que dans les différents algorithmes, afin d'économiser d'avantage d'énergie, et d'être plus proche de la réalité.

La gestion des tâches ne s'arrête cependant pas à la seule allocation statique. En effet, les charges de travail dans un cloud varient au cours du temps. On peut donc avoir d'une part un cycle journalier de la charge, avec plus de charge entre 18h et 22h, mais aussi des pics plus localisés de charge, qui sont difficiles à prédire. Dans tous les cas, il faut prendre en compte cette dynamicité. Pour cela, nous étendrons notre modèle de contraintes linéaires afin de prendre en compte le temps de migration des tâches, le temps d'allumage et le temps d'extinction des machines.

Ensuite, nous avons développé des algorithmes et utilisé un simulateur d'une infrastructure de cloud pour évaluer ces algorithmes, et l'impact de la réduction de l'énergie au niveau de l'allocation des ressources et de la réallocation des tâches. Cette infrastructure nous permettra d'avoir un gestionnaire autonome de tâches, qui, en séparant en trois parties distinctes la reconfiguration des tâches, leur réallocation et la gestion des machines physiques, permettra d'appliquer une approche globale autonome au problème de la gestion efficace des tâches dans un cloud.

Notre approche a montré qu'il était possible de réduire grandement la consommation d'énergie par rapport à une approche non efficace en énergie, tout en gardant un niveau de violation de Service Level Agreement (SLA, qui représente le contrat entre le client et le fournisseur de services et donc la qualité de service) faible.

1.4 Organistion du manuscrit

La suite de ce mémoire est organisée comme suit :

- Le chapitre 2 présente l'état de l'art en matière de gestion de tâches prenant en compte l'énergie dans les clouds. Il présente en particulier les différents leviers matériels et logiciels utilisables dans des techniques de réduction de la consommation d'énergie. Est présenté ensuite un ensemble de techniques développées au dessus de ces leviers afin de gérer les tâches dans un cloud, afin d'analyser ce qui se fait de mieux dans ce domaine et de comprendre ce que l'on peut et doit faire pour avoir une meilleure connaissance des systèmes étudiés.
- Le chapitre 3 présente le système que nous allons étudier et la définission du problème que nous allons prendre en compte et tenter de résoudre. Sont présentées ensuite les différentes hypothèses que nous posons, afin d'étudier certains effets de manière plus spécifique. Nous décrivons formellement les différents problèmes, ainsi que les techniques de résolution des problèmes que nous allons utiliser dans les chapitres ultérieurs.
- Le chapitre 4 présente le contexte du problème de l'allocation statique des tâches dans un cloud, ainsi que les hypothèses et les leviers que nous utilisons dans ce contexte précis. Nous définissons ensuite l'ensemble de contraintes linéaires et les fonctions objectifs qui constitueront le programme linéaire modélisant le problème de l'allocation statique. Une validation par simulation des algorithmes proposés est faite ensuite pour faire émerger les

forces et faiblesses des algorithmes en terme de performance et d'énergie, ainsi qu'étudier l'impact de l'hétérogénéité sur ces deux métriques. Nous étudions aussi l'impact que peut avoir l'infrastructure de refroidissement lorsqu'elle est prise en compte dans l'allocation de tâches. Enfin, nous étudions les relations qu'il peut y avoir entre les différents aspects que nous avons rajoutés, comme la relation entre le refroidissement et l'hétérogénéité des serveurs.

- Le chapitre 5 présente le passage nécessaire de l'allocation statique à l'allocation dynamique. Y sont présentées les raisons qui font que ce passage est nécessaire, ainsi que les différentes contraintes à rajouter au modèle pour pouvoir passer à la dynamicité. Nous évaluons ensuite cette dynamicité avec une simulation d'un gestionnaire autonome de clouds, plus particulièrement l'impact de la reconfiguration de l'allocation effective de chaque tâche, l'impact des algorithmes de réallocation des tâches, la gestion des machines physiques et la possibilité de passer à l'échelle du système.
- Le chapitre 6 enfin conclut ce manuscrit en résumant nos contributions et en présentant des axes de recherches possibles pour affiner ces travaux et poursuivre les recherches.

Chapitre 2

Gestion de tâches efficace en énergie

2.1 Introduction

La gestion de tâches est une discipline largement répandue et qui peut être utilisée dans différents domaines. Elle intervient lorsque l'on possède un ensemble de tâches à accomplir, et que ces tâches sont à exécuter sous contraintes. Ces contraintes peuvent être de différentes sortes. Elles peuvent être des contraintes temporelles, lorsque les tâches doivent être exécutées dans leur totalité avant une certaine échéance. Les tâches peuvent être aussi contraintes par des ressources, quand celles-ci doivent être accomplies avec un budget limité.

Dans le cas qui nous intéresse, en informatique, les tâches sont des processus logiciels qui s'exécutent sur une infrastructure donnée. Comme toutes les tâches ne sont pas identiques, certaines sont finies et d'autres infinies par exemple, elles ne sont pas caractérisées par les mêmes paramètres. Cependant, une caractéristique est commune à toutes les tâches : elles ont toutes des besoins de ressources, que ce soit de manière explicite lorsque la tâche demande un certain nombre de machines ou un accès à des ressources définies, ou implicite lorsque la tâche demande juste la possibilité de s'exécuter, mais aura quand même une consommation de ressources précise.

Si l'on se place dans le monde de l'informatique mobile, nous avons des tâches qui sont faiblement contraintes par le temps, mais fortement contraintes par les ressources qu'elles utilisent, comme le CPU et la batterie que la tâche utilisera.

On peut aussi se placer dans le monde du HPC, où les tâches doivent être exécutées le plus vite possible, et sont donc fortement contraintes par le temps pris à l'exécution, sans se préoccuper de la quantité de ressources utilisées pour arriver au résultat.

Dans un cadre du cloud computing, les tâches peuvent être de différentes sortes. Elles peuvent faire partie d'un même ensemble de tâches, dans le cadre d'une grappe de calcul virtuelle. Dans ce cas, les tâches peuvent être communicantes et, par conséquent être contraintes par l'interdépendance inhérentes aux calculs communicants. Elles peuvent aussi être des services, et devoir être capable de répondre à des requêtes de clients à tout moment.

Dans cette thèse, nous nous intéresserons à des tâches exécutées dans un environnement de type cloud. Ces tâches seront infinies, ce qui veut dire que nous ne travaillerons pas dans des contraintes temporelles. Nous modélisons en fait un environnement avec des tâches de types services web par exemple. Un service web doit toujours être en exécution, ce qui change au cours du temps étant uniquement la charge induite par les requêtes web. Nous serons cependant contraint par la consommation d'énergie de l'infrastructure, ainsi que la consommation de ressources machine demandée (CPU, mémoire, bande passante, disque, etc.).

La gestion de ces tâches s'appuira sur différents leviers (moyens d'actions), et devra résoudre

plusieurs problèmes différents. De plus, celle-ci pourra avoir différentes visées, différents objectifs, et par conséquent être de différente forme. Dans la suite de ce chapitre, nous allons nous attacher à détailler les moyens d'actions, les problèmes à résoudre, ainsi que les objectifs visés.

2.2 Leviers matériels et logiciels

Afin d'agir sur la consommation d'énergie des machines, nous possédons un ensemble de moyens bien différents. Ces moyens sont à différentes échelles, mais chacun a un impact sur cette consommation, et des effets souvent négatifs sur les performances. Ainsi, nous avons des leviers qui vont d'un niveau bas, au niveau processeur ou au niveau machine, à un plus haut niveau, de cluster ou d'infrastructure. Ces leviers peuvent être aussi bien logiciels, comme la virtualisation ou l'allocation, que matériels, comme le DVFS.

2.2.1 Dynamic Voltage and Frequency Scaling

Le DVFS est un mécanisme mis en place par les constructeurs de puces, qui permet de dynamiquement réduire ou augmenter la fréquence d'un composant, ainsi que son voltage. Cela permet de descendre la fréquence d'un processeur, lorsqu'il est sous utilisé, ou au contraire, de l'overclocker (augmenter la fréquence au dessus de sa fréquence maximale) lorsqu'il a besoin d'être plus performant. Ce réglage se fait en réglant un couple voltage/fréquence (appelés "P-State" pour *performance state*), dont les valeurs et effets dépendront du processeur utilisé. Le tableau 2.1 présente un exemple de ces couples et de la consommation de puissance qui leur est associée.

Les différents couples voltage/fréquence sont gérés par le système de la machine. C'est le cas dans les systèmes où sont implémentés des gestionnaires de fréquence CPU que l'on appelle gouverneur, et dont le but est de gérer la fréquence et le voltage du processeur en fonction d'une orientation définie par l'utilisateur, et en fonction de certains paramètres systèmes. Trois gouverneurs différents sont implémentés par défaut : **performance**, **powersave** et **ondemand**. Le gouverneur **performance** règlera la fréquence toujours au maximum, alors que le **powersave** règlera la fréquence toujours au minimum. Le gouverneur **ondemand** quand à lui règlera la fréquence au maximum dès que des calculs à effectuer arrivent, pour la réduire petit à petit quand la charge baisse.

Il faudra noter cependant que bien que souvent un bon nombre de modes soient proposés pour les processeurs, des études montrent que seuls le plus bas et le plus haut sont les plus importants, et qu'un troisième mode intermédiaire n'amène qu'une amélioration faible de la solution optimale dans le cas de la gestion de tâches infinies [88].

TABLE 2.1 – Différents P-states et leur consommation pour le processeur Intel Pentium M 1.6GHz[33].

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

Cependant toute action a un coût, et le DVFS n'y échappe pas et peut ralentir l'application.

Changer de fréquence prend du temps (de l'ordre de la dizaine de microsecondes [28]), et ce temps est passé sans calcul. Par conséquent, des changements trop fréquents de fréquence vont avoir un impact très négatif sur les performances. Un impact négatif sur les performances induira évidemment un mauvais temps de complétion de l'application (si celle-ci se termine), et par conséquent une mauvaise consommation d'énergie. Dans notre cas, puisque nous nous intéresserons à des tâches de type service n'ayant pas de complétion, l'impact négatif sur la performance n'aura pas d'impact sur la consommation d'énergie, mais aura un impact négatif sur la performance de l'application, et donc sur la qualité de service. Il est important de noter qu'un autre inconvénient au DVFS existe. Il peut arriver qu'une application consomme plus en tournant à une fréquence plus basse. Prenons par exemple une application qui se termine en 10 secondes avec un processeur qui consomme 100W à fréquence maximale. Nous aurons donc une consommation d'énergie de 1000 Joules. Admettons maintenant que cette même application consomme à fréquence minimale 50W, mais induit un temps de calcul de 30 secondes, nous avons alors une consommation d'énergie de l'application de 1500 Joules. Par contre, si cette application se terminait en 15 secondes à fréquence minimale, nous aurions une consommation de 750 Joules.

Pour palier au problème de l'augmentation de la consommation d'énergie à basses fréquences, dans [43] les auteurs définissent un *scheduler* (ordonnanceur en français) visant à allouer les tâches HPC sous contrainte d'un budget d'énergie à ne pas dépasser. Ils utilisent les capacités apportées par le DVFS pour régler la fréquence des processeurs en fonction de l'échéance et du ralentissement prédits des tâches. Les résultats observé par simulation sur plus de 4000 processeurs montrent que les algorithmes que les auteurs ont défini apportent une augmentation de performance allant de 20% à 40% pour les tâches en comparaison avec une approche sans DVFS, tout en respectant la borne sur la consommation d'énergie.

Une vitesse d'exécution réduite n'implique pas forcément une réduction de la consommation d'énergie. Ce qui implique que de la même manière, réduire la puissance n'implique pas non plus une réduction de la consommation d'énergie. Un bon exemple est décrit dans [95], où les auteurs présentent et implémentent un algorithme de *scheduling* basé sur des évènements pour le gouverneur de processeur. Pour ce faire, les auteurs ont implémenté un *scheduler* qui, plutôt que d'aller chercher les informations sur la charge, va recevoir des événements en fonction de celle-ci. Ainsi, si la charge était basse et augmente au dessus d'un certain seuil, l'évenement "chargé" va être envoyé, et le gouverneur va ainsi changer la fréquence du processeur.

La comparaison est faite notamment avec les gouverneurs implémentés dans les kernels Linux qui sont **performance** (toujours à la fréquence maximale) et **powersave** (toujours à la fréquence minimale) . On peut voir dans les résultats présentés par les auteurs que le gouverneur **performance** exécute le programme en 1h03 avec une puissance de 310W, duquel résulte une énergie de 329Wh, alors que le gouverneur **powersave** l'exécute en 3h09 a une puissance de 284W, résultant une énergie de 895Wh! Leur algorithme cependant, exécute ce même programme en 1h05 avec une puissance moyenne de 298W, pour une énergie de 324Wh. Cela montre bien qu'il est nécessaire d'utiliser la gestion de fréquence des processeurs à bon escient. Une fréquence plus faible n'économisera pas forcément de l'énergie, et le bon choix sera parfois de favoriser la performance pour économiser de l'énergie avec un temps d'exécution plus court.

Les auteurs dans [60] tentent de palier au problème de l'affectation de la fréquence la plus efficace en énergie en utilisant une caractérisation de la charge de travail par intervalle, pour allouer la bonne fréquence et le bon voltage aux tâches, via un gouverneur visant à réduire la consommation d'énergie en minimisant les pertes en performance, acceptant ainsi des pertes en performances pour avoir de meilleures économies d'énergie. Pour cela, les auteurs caractérisent la charge de travail pour chaque noeud (par opposition à une caractérisation par cluster). Ils définissent ensuite le gouverneur efficace en énergie *ecod*, basé sur les moments où le CPU attend le résultat d'activités hors processeur. Cela permet en effet de pouvoir caractériser la charge en se

basant sur des mesures, et n'avoir besoin d'aucune connaissance des applications qui s'exécutent sur le cluster. L'algorithme est ensuite évalué par l'exécution des benchmarks *NAS Parallel Benchmark* [13], et les auteurs montrent que des économies d'énergie allant jusqu'à 11% peuvent être effectuées par rapport au gouverneur **ondemand**, tout en amenant une perte de performance plus basse (5.1% d'augmentation du temps de calcul contre 7.9% pour ondemand).

Un autre inconvéniant de taille pour le DVFS est l'impact de la fréquence du DVFS sur le fonctionnement du processeur. En effet, des études montrent que l'utilisation du DVFS a un impact négatif sur la fiabilité du processeur [107]. Les auteurs montrent analytiquement que réduire la fréquence à voltage fixe tend à réduire la *performability* (c'est-à-dire la probabilité de finir une application correctement avant échéance lorsqu'il y a des erreurs). De plus, réduire le voltage à des fréquences réduites augmente le taux d'erreurs dans les processeurs.

L'approche de type DVFS n'est pas seulement limitée aux processeurs. C'est une approche qui peut être appliquée aussi à d'autres types de matériels. Par exemple, nous pouvons utiliser une approche similaire au niveau des cartes réseaux. Celle-ci s'appelle l'Adaptative Link Rate (ALR)[53]. Le principe est le même : augmenter ou baisser la vitesse de la carte (son débit, faire fonctionner une carte ethernet 1Gbps à 100Mbps ou moins) en fonction des besoins, afin de limiter la consommation d'énergie. Nous pouvons aussi effectuer le contrôle du voltage et de la fréquence au niveau des GPUs (Graphics Processing Unit), puisque ceux-ci sont utilisés dans certaines architectures hybrides CPU/GPU, où certains calculs sont effectués sur le GPU d'une machine, alors que les autres sont fait sur le CPU. Les GPUs sont de plus traditionnellement plus performants en terme de performance par watts pour les tâches très parallélisables, en en faisant des matériels intéressants pour améliorer l'efficacité énergétique des serveurs comme montré dans [6]. Les auteurs dans [6] montrent de plus qu'identifier les moments où il faut changer la fréquence et le voltage dans une architecture hybride CPU/GPU, permet d'atteindre des économies d'énergie de 28% avec seulement une baisse de 1% en performance dans le cas d'additions et de multiplications de matrices. Enfin, nous pouvons penser aux infrastructures de refroidissement, plus particulièrement aux ventilateurs des climatisations des clusters, qui peuvent tourner à des vitesses différentes, en fonction de la température recherchée[27, 20].

Ce genre d'approche est de plus en plus utilisée. Nous verrons de plus en plus de composants avec différentes vitesses possibles, et nous utiliserons de moins en moins explicitement ces capacités, car les fonctionnements seront directement intégrés, soit au niveau matériel, soit au niveau du système d'exploitation. Le but étant de s'approcher le plus possible d'infrastructures matérielles dites "proportionnelles", c'est-à-dire qui ne consomment qu'une énergie proportionnelle à leur charge[19, 90].

2.2.2 Allumage et extinction des machines

Il existe un moyen simple de réduire la consommation d'une machine. Il suffit de l'éteindre ! Par éteindre, nous entendrons une extinction totale de la machine, pour amener sa consommation à 0W, ce qui est rendu possible par des appareils permettant de "désactiver" les prises de courant [69]. Il faudra bien faire la distinction entre une extinction et une mise en hibernation, qui sont deux états similaires, mais qui ne consomment pas la même puissance. L'extinction de la machine désignera l'arrêt total de tous les composants de celle-ci, réduisant la consommation de puissance à un niveau proche de 0W, alors que ce que nous désignerons comme hibernation l'arrêt du système à l'exception de certains composants utilisés pour la veille du système, comme la carte réseau pour l'utilisation du *wake on lan* (technique consistant à réveiller une machine à distance par le réseau), le *suspend-to-RAM* ou le *suspend-to-Disk*. La figure 2.1 est un exemple de diagramme des différents états d'une machine Linux ainsi que les coûts en temps et en puissance électrique nécessaire pour passer d'un état à un autre. Les valeurs présentées sont tirées de [39].

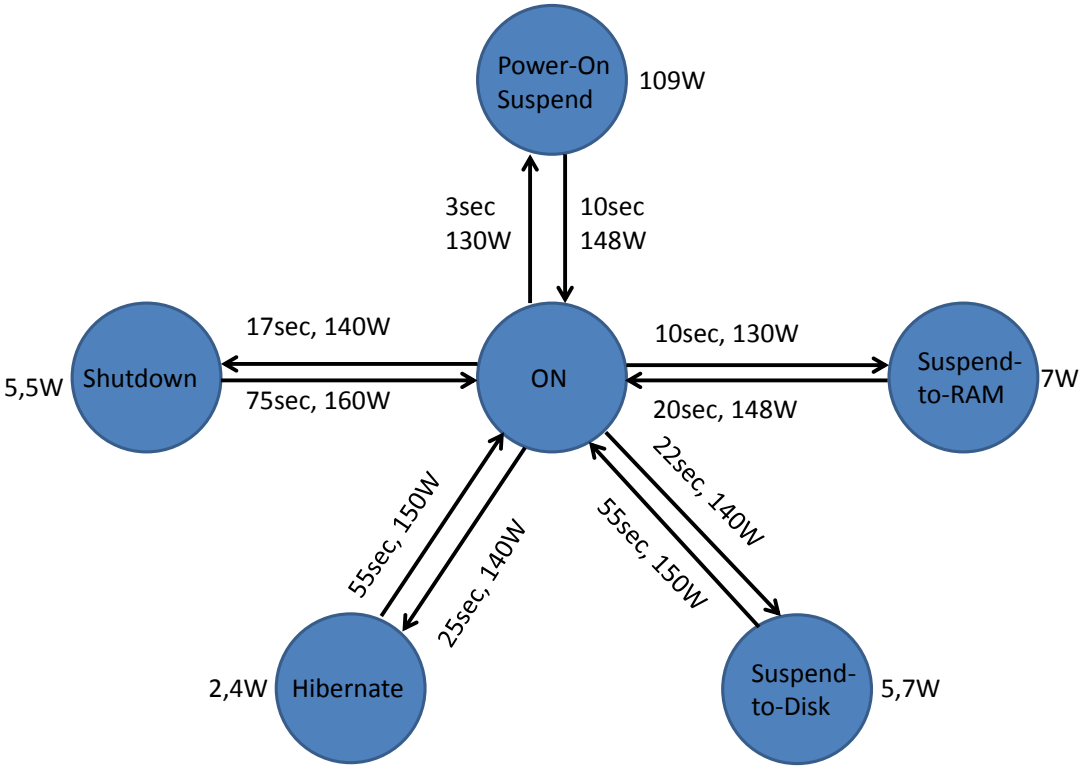


FIGURE 2.1 – Exemple de transition entre états pour une machine Linux équipée d'un AMD Phenom^{TM} 9500 Quad-Core [39]

Ainsi, nous sommes capables de développer des approches utilisant ce levier, afin de réduire la consommation d'énergie. Dans [83], une approche utilisant un processus de décision de Markov est utilisée pour gérer les tâches et l'état (allumé ou éteint) des hôtes, et atteindre l'état optimal pour la consommation tout en préservant la qualité de service comme le temps d'attente des tâches.

Cependant, pour réduire la consommation énergétique d'un server à 0W, nous devons prendre en compte plusieurs facteurs.

Premièrement, nous devons être capables d'éteindre et de rallumer à distance les machines. En effet, dans les clusters actuels, nous n'avons pas tout le temps cette capacité, car les machines ne sont pas faites pour être éteintes. Deuxièmement, nous devons prendre en compte le temps pris pour éteindre et rallumer les différentes machines. Le temps pris par une machine pour éteindre et rallumer les hôtes sera autant de temps pendant lequel les machines ne seront pas utilisables et consommeront de l'énergie. Il faut donc bien prendre en compte le temps d'allumage par exemple, pour ne pas en avoir besoin sur le moment, et ainsi induire une dégradation dans la qualité de service de la tâche. Enfin, il faudra prendre en compte le coût induit par la consommation de ressources lors de l'allumage et de l'extinction. C'est le cas lorsque l'on allume une machine, celle-ci effectue une série de tests matériels et logiciels, en plus du démarrage normal du système d'exploitation. Cela prend donc du temps, mais aussi coûte de l'énergie, dans la mesure où

la machine calcule. Ces temps et consommations d'énergie varient bien sûr en fonction de la machine, du système et des vérifications matérielles qui doivent être faites. Cela peut aller pour le temps d'allumage, de la vingtaine de secondes dans [72] à un peu plus d'une minute comme dans [39], voire même de plusieurs dizaines de minutes comme dans [57].

Toutefois, un effet indésirable pourrait apparaître lorsque l'on sollicite trop l'allumage et l'extinction de matériels. Cela peut en effet induire une augmentation du taux de malfonction de la machine. Bien que rien n'indique qu'au niveau machine le taux de malfonction des hôtes ait un lien avec le nombre de démarrage/extinction [57], ce lien pourrait exister, de la même manière que pour le processeur et le DVFS comme vu dans la section précédente [107].

L'allumage et l'extinction des hôtes physiques est donc un levier important et extrêmement puissant quand il s'agit de réduire la consommation d'énergie d'un système informatique, mais celui-ci est à utiliser avec parcimonie, dans la mesure ou une mauvaise prédiction peut induire de trop fréquents allumages/extinctions, impliquant ainsi un gaspillage de ressource et d'électricité. Il faudra toujours que l'énergie économisée pendant qu'une machine reste éteinte soit plus importante que celle consommée pendant l'allumage et l'extinction, sous peine d'augmenter la consommation d'énergie tout en dégradant la qualité de service.

2.2.3 Virtualisation

Au fur et à mesure que les systèmes deviennent de plus en plus complexes, et les architectures de plus en plus diverses, il apparait clair qu'il faille rendre transparent certains services. Il devient ainsi nécessaire pour faciliter le développement d'avoir un accès commun à des ressources différentes. C'est ici qu'intervient ce que l'on appelle la virtualisation. La virtualisation représente l'idée selon laquelle on rend l'accès identique à plusieurs architectures différentes. Par exemple, on peut avoir un accès virtualisé à des processeurs différents, quel que soit le constructeur de ce processeur, quelle que soit la méthode d'accès à celui-ci.

La virtualisation peut être appliquée à plusieurs choses. Ainsi, nous pouvons avoir la virtualisation du stockage, avec laquelle l'accès aux ressources de stockage sera transparent, quelque soit l'endroit où se trouvent effectivement les disques durs, le type de ceux-cis, et quelque soit le type de ces disques (qu'ils soient de type normal, SSD, mémoire flash...)[97].

Nous pouvons aussi bien avoir une virtualisation de machines, ce que l'on appelle des machines virtuelles. Cela consiste à donner un accès complet à une infrastructure virtuelle, qui apparaitra à l'utilisateur comme une machine complète. Cela peut par exemple servir à donner à deux clients différents accès chacun à une machine complète, sur une même machine physique. Cela peut aussi permettre d'avoir une meilleure sécurité, dans le sens où même si l'on donne un accès administrateur à un utilisateur sur une machine virtuelle, celui-ci ne pourra pas compromettre l'intégrité physique de la machine hébergeant la machine virtuelle [17].

Les machines virtuelles ont aussi un avantage conséquent qui est celui de la migration. En effet, les technologies actuelles de virtualisation permettent de migrer une machine virtuelle d'une machine physique à une autre. Cela peut se faire de deux manières. La première est une méthode qui consiste à mettre en pause la machine, la migrer et la reprendre une fois la migration terminée. La deuxième manière permet une migration dite "live". Il est possible de migrer une machine virtuelle presque sans perturber son fonctionnement, c'est-à-dire sans arrêter son fonctionnement mais avec une interruption de service minimale pouvant aller de 60ms pour un serveur de jeu à faible latence à 210ms pour un serveur avec un workload de type SPECweb99[2]. Ce type de migration permet d'avoir une continuité du service, avec juste une perturbation minimale au niveau réseau, lors de la dernière partie de la migration [32].

Il existe plusieurs implémentations de la migration "live" de machines virtuelles. L'approche traditionnelle, qui est celle qui est implémentée dans Xen ou KVM par exemple, est dite en

pré-copie. Le processus de migration commence par une phase de pré-copie de la mémoire de travail d'une machine virtuelle, puis va itérativement transférer à nouveau les pages qui ont été utilisées par la machine virtuelle pendant le transfert précédent, ceci jusqu'à obtention d'un noyau de pages mémoire suffisemment petit, ou jusqu'à ce qu'un nombre prédéfini d'itération ait été effectué. La machine virtuelle est ensuite arrêtée et le noyau de mémoire final restant à transférer est envoyé sur la machine virtuelle destination, qui est démarée à son tour, marquant la fin de la migration.

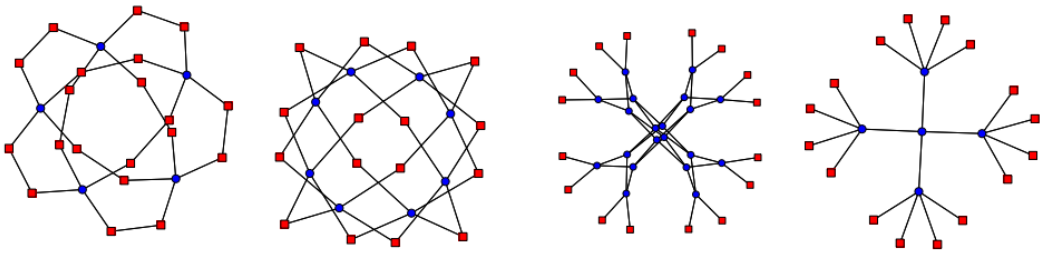
D'autres approche ont été étudiées, comme Hines et al. dans [58], où les auteurs présentent l'implémentation et l'évaluation d'une migration live basée sur la post-copie de la mémoire. Le principe est ici de transferrer l'état processeur en premier et lancer la machine virtuelle destination, puis aller chercher sur la machine source les pages mémoires qui font défaut. Cette technique, contrairement à la pre-copie permet de s'assurer que les pages mémoires d'une machine virtuelle donnée sont transférées au plus une fois. Par conséquent cette technique permet de réduire grandement le volume des données à transférer lors d'une migration, mais augmente aussi les latences d'accès aux données lorsqu'une page mémoire doit être récupérée à distance. Les auteurs montrent dans leurs expériences avec des applications réelles, que la post-copie, bien qu'augmentant très fortement l'interruption de service (plus de trois fois dans certains cas), permet de réduire le nombre de pages transférées de 28% pour le workload le plus lourd (SPECweb), de plus de 50% pour les autres, allant jusqu'à 72% pour le meilleur. De plus, la post-copie permet de réduire le temps de migration d'au moins 23% pour le benchmark SPECweb, et jusqu'à 62% pour une compilation de noyau Linux. Des approches hybrides pre/post-copie sont aussi envisagées dans le futur par les auteurs, toujours dans le but d'accélérer la migration, en effectuant une seule itération de pre-copie avant de transférer l'état CPU et de passer à une étape post-copie.

Enfin, les machines virtuelles ont l'avantage de permettre une séparation logique des ressources de la machine. Par exemple, on pourra allouer deux machines virtuelles sur un même hôte, en allouant par exemple 70% du CPU à la première machine virtuelle, et les 30% restant à l'autre. Il apparait donc que la gestion des tâches est facilitée par les technologies de virtualisation, notamment au niveau du provisionnement des tâches. Dans [59], les auteurs utilisent le fait que les machines virtuelles ne consomment pas tout ce qui leur est attribué pour surprovisionner les machines physiques, afin de maximiser leur utilisation effective, et ainsi dans l'exemple que les auteurs prennent, d'économiser 20% de ressources.

Il y a bien sûr des inconvénients à ces techniques de virtualisation. La technique de virtualisation elle même induit une baisse de performances. La gestion transparente des ressources étant gérée par un intergiciel, celui-ci engendre des coûts de calculs et de gestion des ressources. Ainsi, un des inconvénients majeurs de la virtualisation est le surcoût associé à celle-ci [37]. Le surcoût est notamment dû au fait que d'une part l'accès aux ressources matérielles (GPU, disques durs, etc...) est plus lent du fait de l'intergiciel de virtualisation qui doit faire le lien entre le système hôte et la machine physique, et du fait que les défauts de page mémoire sont significativement plus coûteux. Ce surcoût dépend d'un nombre de paramètres tels que le programme et l'architecture matérielle qui est virtualisée, et peut atteindre lorsque le cache est trop petit des valeurs de ralentissement telles que 17% pour une architecture Intel et 38% pour une architecture AMD [37].

D'autres inconvénients sont liés à la migration de machines virtuelles. La migration de machines virtuelles nécessite un système de fichiers partagés qui héberge les images systèmes et les données disque dur, afin de limiter le volume de données transférées lors de la migration. Cependant, les données à transférer restent importantes, et nécessitent un travail de la part des machines source et destination. La migration induit donc une consommation de ressources à la fois sur l'hôte source et sur l'hôte destination. Il faudra aussi préter attention aux goulots

d'étranglement au niveau des transferts réseaux qui peuvent apparaître si l'on effectue plusieurs migrations à la fois, et plus généralement pour tout gros volume de transfert. L'architecture d'une structure hébergeant une infrastructure virtualisée sera donc importante pour éviter, ou en tout cas limiter, ce genre de problèmes. Nous pouvons en effet utiliser la topologie du système pour effectuer des économies d'énergies en limitant les goulots d'étranglement qui peuvent apparaître lors de trop nombreuses migrations, ou transferts de données en général [55]. La figure 2.2 présente les différentes topologies de data centers étudiées par les auteurs dans [55]. L'infrastructure *Balanced Tree* est celle qui donne les meilleures performances énergétiques, mais est aussi celle qui à le plus de risque de goulot d'étranglement, en particulier le noeud racine. La topologie *Fat Tree* par contre donne des résultats corrects en terme de consommation de puissance électrique (29.8kW pour 3456 serveurs), mais possède la caractéristique d'avoir moins de risques de goulots d'étranglement, chose importante lorsque l'on conçoit un data center dont les machines virtuelles feront objet de beaucoup de migrations.



(a) $\text{DCell}_4(1)$ (b) $\text{BCube}_4(1)$ (c) Fat-tree(4) (d) Balanced tree$(4, 2)$

FIGURE 2.2 – Différentes topologies de data center étudiées dans [55]

Enfin, cette migration d'une machine physique vers une autre possède un coût non négligeable, qui entraine une consommation de ressources sur à la fois l'hôte source et destination, ce qui augmentera la consommation d'énergie pendant la migration. De plus, les dernières phases de copie de la mémoire lors de la migration, juste avant de terminer celle-ci peuvent entrainer sur certaines machines une dégradation de la qualité de service. Par exemple, dans [104], les auteurs étudient le coût de la migration dans les clouds, avec des applications de type services web. Il apparait que lors de la migration, le temps de réponse peut dans certains cas monter jusqu'à 3 secondes, induisant ainsi des violations de SLA.

Le coût de migration peut être pris en compte dans la décision, afin de ne pas faire de réallocations de machines virtuelles contre performantes. C'est le cas dans GreenCloud [77], une architecture de gestion de machines virtuelles. Il y est défini une heuristique de replacement performante prenant en compte le coût de la migration, le coût de la consommation d'énergie, (ici modélisé par le nombre de machines allumées), ainsi que le coût de la qualité de services (modélisé par le nombre de serveurs avec une utilisation supérieure à 90%).

Pour palier à la mauvaise performance des migrations Al-Kiswany et al. présentent dans [10] VMFlockMS, un service de migration de machines virtuelles performant et passant à l'échelle. VMFlockMS tire partie de la caractérisation pour effectuer de la co-migration, et tente d'être performant lors de migrations inter clusters. Le service de migration tire aussi partie du fait que l'image des machines virtuelles qui sont migrées en même temps, et font partie de la même application auront une grande similarité, et copier chacune entièrement ne sera pas forcément nécessaire. De plus, il arrivera souvent des cas où les images sources auront des similitudes avec des machines déjà disponibles sur le cluster de destination, limitant ainsi le volume nécessaire

à transférer. Enfin, les auteurs transfèrent certaines parties de l'image en premier, les parties nécessaire au démarrage de la machine virtuelle, afin de pouvoir la démarrer au plus vite, et ainsi gagner du temps de migration. VMFlockMS est ainsi capable d'accélérer le transfert de machines virtuelles inter centres de calculs jusqu'à 3.5 fois, ainsi que de réduire la taille transférée à 3% de la taille initialement transférée par VMFlock.

2.2.4 Allocation en tant que moyen/levier

L'allocation de tâches est aussi un moyen de gérer la consommation d'énergie d'un système. Ainsi, en plaçant de façon intelligente les tâches, nous pouvons utiliser les infrastructures les moins gourmandes en électricité.

C'est le cas des techniques dites de consolidation. Le but étant de regrouper la charge de travail au même endroit pour pouvoir réduire la consommation d'énergie. Consolider les tâches sur un nombre réduit de machines peut avoir des effets néfastes. Outre le fait que l'on puisse tenter de consolider sans succès (c'est-à-dire tenter de consolider sur un nombre réduit de machine, mais ne pas réussir, et donc utiliser trop de machines), il faut aussi prendre en compte les interactions que peuvent avoir les tâches entre elles lorsqu'elles s'exécutent sur une même machine. Dans [99] les auteurs montrent en profilant la consommation de ressources d'une tâche qui s'exécute parmi d'autres, la consommation de ressources et l'énergie consommée par chaque transaction de celle-ci augmentent comparativement au cas où elle est seule. De plus, l'efficacité énergétique d'une transaction tend à augmenter aussi si la tâche s'exécute toute seule sur la machine. Les auteurs montrent ensuite que l'on peut trouver une zone optimale à laquelle chaque tâche s'exécutant sur la machine consommera le moins d'énergie. Ils proposent ensuite un algorithme de consolidation basé sur le profil des interactions des tâches, pour déterminer la consolidation optimale ou proche de l'optimale, ainsi que pour caractériser quels types de charges de travail doivent être combinées.

L'allocation peut aussi être utilisée en tant que levier dans la mesure où une caractérisation des différentes ressources matérielles peut faire une différence importante. Tous les serveurs ne consomment pas la même puissance pour exécuter la même application, et utiliser le levier de l'allocation des tâches s'avère donc important. Différentes techniques peuvent être observées pour prendre en compte les différentes consommations à différents niveau, que ce soit au niveau du processeur ou au niveau de la machine elle-même.

C'est le cas par exemple de Dupont et al. dans [38], où les auteurs utilisent l'allocation et la réallocation des machines virtuelles pour mettre en oeuvre un *framework* (ensemble de composants logiciels d'une architecture logicielle) flexible et efficace en énergie. Pour ce faire, les auteurs basent la gestion des clusters par programmation par contraintes sur Entropy [56] afin de pouvoir être flexible en exprimant les différents objectifs pour les algorithmes. Ainsi, cela donne la possibilité d'ajouter ou d'enlever des contraintes pour exprimer les contraintes SLA sans changer les algorithmes. Les auteurs valident ensuite leur *framework* par simulation et expérimentations. Ils montrent que dans un environnement cloud de test, leur approche permet de réduire jusqu'à 18% d'émissions CO2.

2.2.5 Complexification de l'allocation de tâches

À la lumière de tous les moyens d'action dont nous disposons, il apparait que l'allocation de tâches a évolué vers quelque chose de bien plus complexe. Allouer des tâches n'est plus juste décider où allouer une tâche sur un ensemble de machines, mais c'est aussi la gestion de ces machines, en prenant en compte la totalité des moyens auxquels nous avons accès après l'allocation. Il faut maintenant prendre en compte aussi les différentes fréquences dont on dispose sur les différents hôtes, la consommation électrique des ces hôtes, prédire les besoins dans le temps,

afin d'avoir une provision suffisante de machines allumées.

Il faut ajouter à ceci le fait qu'il y a une différence d'échelle dans les différents leviers, ce qui implique une gestion potentiellement différente de ceux-ci. Par exemple le DVFS s'effectue au niveau du processeur, alors que l'allumage et l'extinction se fait au niveau des hôtes. Le refroidissement doit être géré au niveau cluster, alors que la migration se fait au niveau des hôtes. Nous avons donc une approche multi-niveaux, qui doit prendre en compte tout le spectre des approches possibles pour réduire la consommation d'énergie, du niveau cluster au niveau composant.

Cependant, ces nouvelles contraintes rendent le problème de l'allocation de tâches d'autant plus complexes que certains leviers induisent des effets qui peuvent avoir des interactions antagonistes.

Prenons l'exemple du DVFS. Nous voulons baisser la fréquence d'un hôte, ce qui induit une baisse de la consommation de cette machine. Cependant, nous pouvons facilement imaginer qu'il y aura d'autres tâches qui auraient pu être exécutées sur cet hôte, à condition que celui-ci tourne à pleine vitesse, ce qui aurait pu permettre d'éteindre un deuxième hôte, et ainsi de réduire encore plus la consommation d'énergie du système.

On pourra aussi facilement penser au cas où la consolidation de machines virtuelles sur une machine physique aura un effet indésirable sur le provisionnement des machines virtuelles (dans le cas par exemple d'une consolidation trop aggressive des machines virtuelles). Ainsi, si beaucoup de machines virtuelles sont allouées sur une seule machine physique, il sera difficile d'augmenter la part de ressource allouée aux machines virtuelles, puisque peu de ressources resteront libres, cela sans compter sur l'interférence inter machines virtuelles lorsqu'elles sont sur la même machine physique [81].

Par contre, il est important de noter que même si les différentes échelles de leviers de réduction de la consommation d'énergie via gestion des tâches complexifient le problème, ignorer certaines des interactions ne voudra pas dire que les résultats ne seront pas bons. Si l'on prend les travaux de Chen et al. dans [31], les auteurs présentent une solution de gestion de data centers comprenant des composants de gestion de trois niveaux : application, noeud et groupe de noeuds. Ils montrent par expériences que leur prototype de solution intégrée permet de réduire l'énergie des serveurs de 35% et celle du refroidissement de 15% sur une expérience de 10h comparé à un placement statique, et ce sans dégrader la performance des applications.

2.3 Placement de tâches contraint

Le placement de tâches peut être utilisé de diverses manières. En effet, on peut utiliser les mêmes techniques pour optimiser un système en fonction d'un ou plusieurs objectifs différents. Il en va de même pour le type de systèmes, ainsi que ce sur quoi le ou les algorithmes de gestion vont s'appuyer. Tout dépendra des contraintes que l'on prendra en compte pour la gestion de notre système.

L'idée sous-jacente est d'avoir une gestion de ressources qui se fera en fonction de certains critères. Cela peut aller d'un coût monétaire pour maximiser les profits, à l'utilisation d'un certain type de ressource comme l'utilisation d'énergies renouvelables. Nous pouvons aussi très bien avoir des considérations internes aux applications qui s'exécutent sur l'infrastructure physique quand par exemple les tâches sont fortement contraintes en mémoire, ou contraintes en temps par des échéances.

2.3.1 Placement contraint en ressources

La gestion de tâches contrainte en ressources que nous appellerons *resource-aware*, c'est à dire avec considération de ressources, représente l'idée selon laquelle nous possédons un ensemble de ressources, par exemple de l'électricité de différent types, et ces ressources ne sont pas accessibles de partout, en tous cas pas de la même façon. La gestion de tâches *resource-aware* donnera alors à chaque tâches un endroit où s'exécuter, afin par exemple de consommer le plus possible d'énergie renouvelable. L'idée ici est d'optimiser la consommation d'électricité globale, dans une direction particulière, qui est celle de l'énergie renouvelable. C'est le cas dans [48], où l'on va allouer les tâches tirant partie de l'hétérogénéité qui existe entre les différents clusters d'un cloud pour allouer des tâches HPC selon différentes métriques comme les émissions CO2, le coût de l'électricité, ou l'efficacité des processeurs.

Cependant, ceci n'est qu'un exemple de ce que l'on peut avoir comme situation. Nous pourrions tout aussi bien avoir un ensemble de clusters géographiquement distribués, et vouloir effectuer une gestion de tâches prenant en compte cette localisation, afin de rapprocher autant que possible une tâche de son utilisateur, pour en améliorer la qualité de service [63]. Ce sera souvent le cas dans les réseaux de livraison de contenu, où l'on a besoin, du fait du volume des données à transférer, de réduire autant que possible la distance entre le client et le serveur.

D'une manière générale, le placement de tâches contraint en ressources permet d'entrainer l'allocation résultante dans une direction précise, en fonction des besoins. Ainsi, l'allocation devra résoudre les différents problèmes liés aux différentes contraintes possibles.

2.3.2 Placement contraint en énergie

Le placement contraint en énergie est en lien avec la gestion de tâches *resource-aware*, dans la mesure où l'on peut le définir comme un placement contraint en ressource, avec comme ressource l'énergie consommée ou à consommer. La gestion de tâches peut ainsi tenter d'optimiser à plusieurs niveaux la consommation d'électricité de l'infrastructure. Nous référons à cette allocation de tâches comme *power-aware* ou *energy-aware* (rappelons que le *power-aware* raisonne sur la puissance électrique instantannée, et donc optimise à un instant donné, alors que l'*energy-aware* raisonne sur l'énergie consommée sur une durée, donc l'intégrale de la puissance instantanée). Dans ce cas ci, les techniques mises a notre disposition à tous niveaux sont mises en oeuvre pour réduire l'énergie consommée par l'ensemble des machines. Nous pourrons utiliser tous les leviers mis à notre disposition afin de réduire cette consommation.

L'ensemble des techniques d'allocation de tâches *power-aware*, c'est à dire avec considération de puissance, n'utilisera pas forcément tous les leviers mis à disposition puisque cela rendra la gestion des tâches plus compliquée. Cependant, un point commun à toutes ces approches, les décisions seront basées sur la consommation d'électricité des machines.

À partir de ce constat, nous aurons des approches power-aware à chacun des niveaux. Par exemple au niveau des composants, nous utiliserons le DVFS pour réduire la consommation des tâches individuellement. Cela veut dire réduire la fréquence du processeur pour réduire sa consommation. Certaines approches *energy-aware* prennent même en compte une possibilité d'overclocking du processeur, dans les cas où accélérer la terminaison d'une tâche réduira sa consommation d'énergie [42]. Il y a en effet deux moyens de réduire la consommation d'énergie via DVFS. Le premier consiste à utiliser le DVFS pour finir plus vite la tâche, le second à finir au même moment en s'exécutant à une fréquence plus basse, et donc consommer mieux les ressources.

C'est le cas lorsque l'on effectue une allocation en gérant des groupes de tâches pour un programme modélisé par un graphe orienté acyclique. Ce sera le cas lorsque nous aurons des tâches soit communicantes, soit pour modéliser un programme en plusieurs phases. Autrement

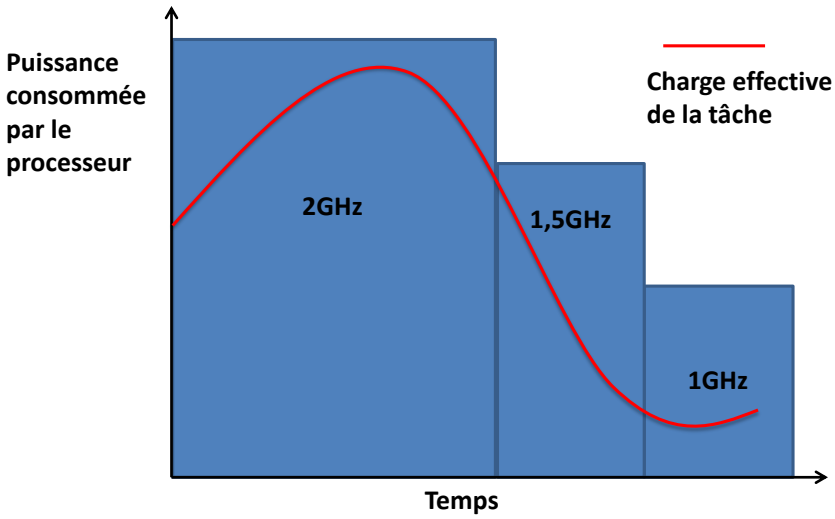


FIGURE 2.3 – Un exemple d'utilisation du DVFS

dit, chaque tâche devra attendre la terminaison de la tâche qui la précède dans le graphe. Avec ce type de modélisation, il est possible de définir des algorithmes pour gérer l'allocation de tâches avec DVFS pour réduire la consommation d'énergie sans pour autant augmenter le temps de complétion global du programme [105, 71]. Cela sera atteint principalement en réduisant la fréquence lorsque l'on attend la fin d'une tâche, ou d'une partie de tâche.

Si l'on accepte par contre une perte de performance raisonnable, nous pouvons atteindre une bien meilleure performance en termes d'énergie, comme dans [96], où les auteurs définissent des algorithmes pour l'ordonnancement de tâches représentées par des graphes acycliques orientés afin d'avoir un compromis entre performance et économies d'énergies, et sont capables en prenant en compte ces deux métriques d'améliorer significativement l'efficacité énergétique de leur système hétérogène (les machines du cluster n'ont pas toutes les mêmes capacités), tout en gardant une augmentation de temps d'exécution acceptable.

Nous avons aussi des approches au niveau machine. Dans ce cas là, il convient de gérer le placement des tâches sur les différentes machines, en fonction de leur consommation. Il faudra aussi gérer l'allumage et l'extinction des hôtes, afin de réduire la consommation globale du cluster.

Dans [87] est développé une stratégie d'optimisation basée sur une formulation en programme linéaire mixte entiers et rationnel (MILP, *Mixed Integer Linear Programming*), afin de consolider dynamiquement les services dans un cluster virtuel. Cette approche prend en compte le coût d'allumage et d'extinction des machines et est utilisée dans le cadre d'un système surdimensionné

afin de pouvoir subvenir aux besoins des pics de charge, amenant ainsi une réduction de 50% de l'énergie consommée dans le cluster.

Une approche similaire est développée par Viswanathan et al. dans [103], où les algorithmes d'allocation de machines virtuelles sont basés sur un modèle empirique (benchmarks sur différents types d'applications comme orienté CPU, orienté E/S, etc...). Un compromis est fait entre l'énergie et le *makespan*(temps entre le début et la fin d'une série de tâches), et arrive ainsi à économiser 12% d'énergie tout en réduisant le *makespan* de 18% en moyenne par rapport à une approche standard de type *first fit*(algorithme glouton où chaque tâche est allouée au premier hôte sur lequel elle rentre).

Lim et al. dans [75] prennent une approche permettant d'utiliser au maximum les ressources disponibles aux applications en utilisant les leviers de DVFS et de partage de temps CPU. Leur application permet d'avoir un budget d'énergie pour chaque différente application. Cela permet de pouvoir répondre à des scénarios tels qu'un système surchargé, ou l'utilisation de data center avec des prix d'électricité variable. Les auteurs valident ensuite leur application avec l'exécution d'un benchmark imitant l'exécution d'un site web d'application boursière, de benchmarks de la suite SPEC CPU 2006, ainsi que de traces d'exécution d'une application de messagerie instantanée. Les résultats apportés améliorent la consommation d'énergie entre 4% et 37% en fonction des controlleurs et des applications regardés, en comparaison à un système sans gestion.

2.3.3 Placement contraint en coût

La contrainte en coût peut vouloir dire deux choses. Cela peut vouloir dire allouer les tâches de manière à réduire le coût de maintenance pour le propriétaire de l'infrastructure. Cela peut aussi vouloir dire gérer de manière à réduire les coûts d'exécution des tâches, afin d'améliorer la qualité de service. Pour réduire ces coûts, il faut d'abord les comprendre, car ceux-ci peuvent être compliqués à prendre dans leur globalité. Dans [12], les auteurs tentent d'évaluer l'impact des data centers en terme de coûts énergétiques et monétaires en évaluant l'énergie consommée et le matériel utilisé. Ils montrent ainsi que les installations plus larges ont souvent un empreinte carbone plus faible sur chaque aspect (électricité, infrastructure de refroidissement, serveurs) que les plus petits data centers, et que les data centers de type conteneur (les serveurs, l'alimentation et le refroidissement sont dans un conteneur de taille standard) ont une consommation d'énergie opérationnelle plus faible.

Dans [40] au contraire, l'approche observée est celle de la maximisation du profit pour les fournisseurs de services, en utilisant les économies d'énergies via l'allocation des requêtes clients. Cette approche nécessite l'implémentation de politiques pour des *workloads* (charges de travail) dynamiques, en prenant en compte notamment les erreurs dues à la prédiction de la charge de travail. La première politique est une politique qui maximise le revenu en calculant le nombre optimal de serveurs à allumer, en prenant en compte le coût de changement d'état, ainsi que le revenu généré par chaque tâche. Une politique basée sur une heuristique appelée QED (Quality and Efficiency Driven) permet de calculer une valeur sous optimale du nombre de serveurs à allumer pour répondre à la demande. Cette heuristique prend notamment en paramètre une valeur α permettant de décrire une évaluation empirique de certains paramètres du système, comme la probabilité que tous les serveurs soient chargés par exemple.

2.3.4 Placement contraint en température

L'aspect thermique est intimement lié à celui de la consommation d'énergie. Ainsi, si l'on parvient, par le biais de la gestion de tâches à réduire la température des machines, deux choses vont en découler. Premièrement les machines individuellement consommeront moins d'énergie,

puisque la charge processeur sera moins importante, ce qui impliquera une baisse de la consommation due au refroidissement propre à la machine. Deuxièmement, nous aurons un refroidissement réduit de l'infrastructure complète, et par conséquent un coût de refroidissement moins important. Enfin, plus il y a de chaleur dans les composants, plus ceux-ci consommeront de la puissance.

La gestion *thermal-aware*(c'est-à-dire avec considération de température) consiste tout simplement à allouer les tâches de manière à éviter les points chauds dans l'infrastructure. Autrement dit, il faut équilibrer la chaleur émise par les machines, afin de réduire la chaleur globale du cluster. C'est le cas dans [31], où l'algorithme de placement tente de faire s'exécuter les tâches là où elles seront exécutées aux endroits refroidis le plus efficacement. Cela permet une réduction de la consommation d'énergie des serveurs de 35%, ainsi qu'une réduction de la consommation de l'infrastructure de refroidissement de 15% par rapport à une approche fixée, et ce sans impacter la performance des applications.

Un exemple de ce genre de technique peut être par exemple les travaux de Banerjee et al[15]. Ceux-ci effectuent un placement de tâches en prenant en compte à la fois le refroidissement, et la température, obtenant ainsi une réduction de la consommation d'énergie allant jusqu'à 15% comparé à une approche efficace en énergie sans éteindre les hôtes inutilisés. La réduction de la consommation d'énergie sera par contre de 9% comparée à une approche efficace en énergie qui utilise l'extinction des hôtes. Le débit de tâches par unité d'énergie quand à lui voit une amélioration allant jusqu'à 6.89%.

Dans CoolIT [82], un gestionnaire de charge intelligent, la gestion de la charge des serveurs et du refroidissement est faite de façon coordonnée pour améliorer l'efficacité globale de l'infrastructure, des serveurs et des équipements de refroidissement. La gestion coordonnée fonctionne ainsi : le composant gérant les machines virtuelles définit la charge désirée, basée sur la considération des applications et des considérations d'hétérogénéité. Le composant gérant les CRACs définit un budget de puissance pour les serveurs basé sur la vitesse minimale du CRAC supportant cette charge. Les résultats par simulation montrent que des performances plus hautes peuvent être atteintes avec une telle gestion coordonnée, et une performance de 18% plus haute peut être atteinte pour une vitesse de CRAC donnée. Ces résultats montrent bien, comme nous allons le faire par la suite, la nécessité d'avoir une gestion coordonnée entre l'infrastructure de refroidissement et la gestion des tâches.

Dans [70], les auteurs étudient la gestion de la température dans un centre de données de cloud HPC. Ils introduisent tout d'abord une nouvelle notion, celle du déséquilibre en chaleur, qui représente l'idée de la différence entre la génération de chaleur et son extraction en différents points du data center. Les auteurs définissent ensuite VMAP, une solution de consolidation efficace en température prenant en compte cette notion de déséquilibre de chaleur. Le but est ici de maximiser l'utilisation de ressources tout en minimisant la consommation d'énergie du data center et en optimisant l'extraction de chaleur. Les résultats sont ensuite validés avec des traces d'exécution d'application HPC. La prise en compte de l'extraction de chaleur et de la charge induite de la solution VMAP permet d'être 9% plus efficace en énergie qu'une approche de type *best-fit* (chaque tâche est allouée là où il restera le moins de place après son allocation), et jusqu'à 35% plus efficace qu'une approche de type *cool-job*, qui sont deux approches traditionnellement utilisées dans le domaine.

2.3.5 Placement avec contrainte temporelle

Lorsque l'on introduit la notion de temps dans le placement de tâches, c'est à dire début et terminaison de celles-ci, nous passons dans ce que l'on appelle le *scheduling* (ordonnancement en français). En effet, le *scheduling* désigne le placement et l'exécution de tâches dans le temps. Cela

veut dire tout d'abord que nous choisirons non seulement où s'exécuteront les tâches, mais aussi à quel moment elles s'exécuteront. Cette approche est celle qui est privilégiée dans le monde du calcul haute performance, du fait de son adéquation avec la finition des tâches. Dans le monde du HPC, les tâches représentent souvent des calculs, qui par nature doivent se terminer dans un temps raisonnable. C'est-à-dire par exemple prédire la météo des jours à venir qui doit être bien évidemment terminée avant le jour prédit. Cependant, les clusters HPC peuvent héberger plusieurs simulations ou programmes distincts au même moment. C'est ce qui rend nécessaire une allocation temporelle en fonction de l'arrivée des différentes tâches. Ainsi, les deux métriques à considérer deviennent l'énergie plutôt que la puissance électrique, et le temps d'exécution des tâches. [49]

Cette approche temporelle implique l'introduction d'un outil particulier qui est la file d'attente. Lorsque l'on dispose de tâches que l'on peut exécuter à différentes vitesses, ce qui veut dire que l'on a la capacité de les retarder ou non, on peut aussi se garder la capacité de les retarder dans la file d'attente. Gérer cette file d'attente sera partie intégrante du *scheduling energy-aware*. En effet, il arrivera souvent des cas où en attendant juste un peu qu'une place se libère sur un hôte nous pourrons éviter d'allumer une machine, ou éviter d'en maintenir une allumée, et par conséquent réduire la consommation d'énergie du système. Cependant, nous voyons bien ici que le compromis n'est pas simple : parfois la réduction de l'énergie du système entrainera une augmentation de la durée d'exécution, au détriment donc de l'exécution de la tâche. La figure 2.4 montre un exemple d'ordonnancement. Nous pouvons y voir qu'exécuter les tâches dans l'ordre d'arrivée (rouge, verte, bleue) donne un temps d'exécution plus grand, et donc une consommation d'énergie plus haute que de retarder l'exécution de la tâche verte pour commencer l'exécution de la tâche bleue au plus tôt.

Le *scheduling* s'appuie aussi sur la prédiction de l'arrivée des tâches, ou plus généralement la prédiction de la charge future des serveurs, pour réduire la consommation d'énergie. Il est effectivement intéressant de tenter de prédire la charge future pour pouvoir anticiper les variations de charge du système et ainsi répondre en adéquation avec les besoins. Cela implique que nous pourrons réduire l'énergie de façon plus aggressive quand nous verrons une baisse de charge arriver, ou faire le contraire lors d'un pic de charge. Dans [22], les auteurs utilisent des techniques d'apprentissage (*machine learning*) afin de prédire la charge CPU et la consommation de puissance des machines, afin d'améliorer la prise de décisions pour le scheduling.

Dans [85], les auteurs introduisent des politiques de gestion de tâches et de machines dont le choix est basé sur des estimations de la consommation d'énergie des tâches et des machines en fonction du moment où les tâches vont être exécutées. Ces politiques sont implémentées dans une infrastructure de réservation pour les tâches, afin de réduire la consommation d'énergie en tirant partie de la sous utilisation de l'infrastructure pour éteindre les noeuds inutilisés. Les auteurs montrent ici que les politiques présentées auraient permis de réduire la consommation du site Grid5000 de Lyon en 2007 de 73800 kWh (toutes les machines étant considérées toujours allumées et consommant 190W).

2.4 Approches réactives

Contrairement aux approches proactives de placement présentées précédemment, les approches réactives ne vont pas tenter de réduire la consommation d'énergie en placant les tâches avant leur exécution, mais plutôt, une fois les tâches en cours d'exécution, décider d'actions pour accomplir différents objectifs.

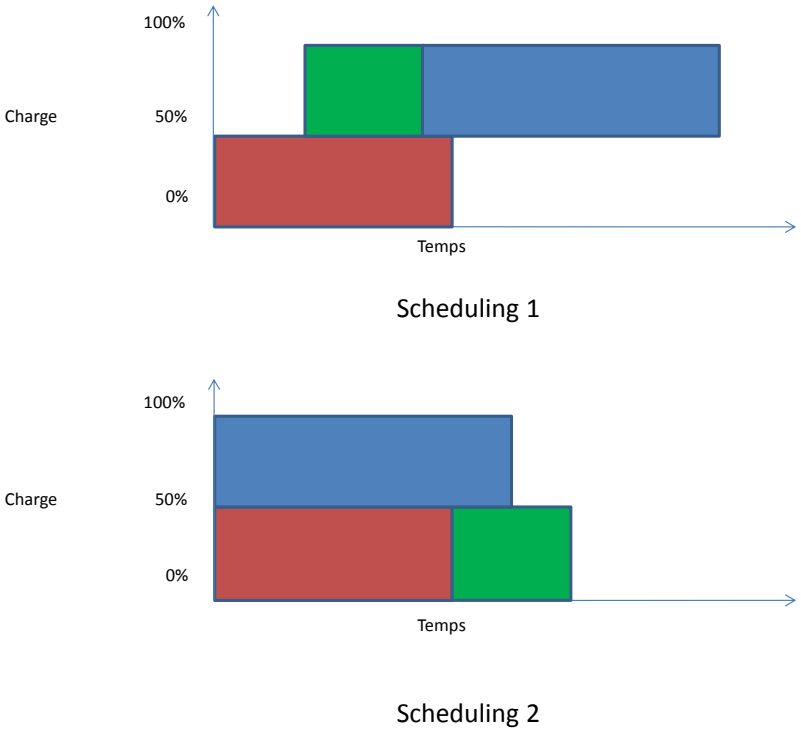


FIGURE 2.4 – Un exemple d'ordonnancement

2.4.1 Approches virtualisées, provisionning et migration

Bien que beaucoup d'approches soient basées sur la virtualisation des tâches, ou l'utilisation de systèmes virtualisés plus généralement, certaines approches se focalisent sur un aspect particulier de la virtualisation. C'est le cas du provisionnement de machines virtuelles. Cette technique consiste à gérer la création et la provision des machines virtuelles pour répondre aux différents besoins des utilisateurs. Ainsi dans [92], l'approche consiste à maximiser l'utilisation de ressources, c'est-à-dire minimiser les coûts de sur-provisionnement et de re-provisionnement (les coûts liés au fait que l'on change trop souvent de nombres de machines), tout en réduisant la consommation d'énergie en éteignant les hôtes. L'approche consiste à inclure chaque tâche qui arrive dans un groupe de tâches ayant des caractéristiques similaires, ou dans un groupe à part si elle n'appartient à aucun groupe. Ensuite, chaque tâche est allouée à une machine virtuelle préalablement décrite en fonction des caractéristiques du groupe. Si une machine virtuelle est disponible du fait d'une tâche terminée, elle est utilisée pour réduire les coûts de reprovisionnement. Enfin les groupes de machines virtuelles sont provisionnés au plus près de ce qu'elles avaient requis. Cela est fait en combinaison avec une modélisation de la charge de travail, afin de mieux profiler le comportement des groupes de machines virtuelles. Leur approche permet d'économiser en moyenne 15% d'énergie, tout en gardant une baisse acceptable de 5% de qualité de service.

Avec le même objectif dans [80], les auteurs tentent de maximiser l'utilisation des ressources

des machines virtuelles. Un service tournant dans une machine virtuelle ne consommera pas forcément tout ce qui lui aura été attribué par SLA, et par conséquent, on pourra réduire cette portion de machine physique allouée à une VM sans impacter sur le fonctionnement du service et sans que celui-ci ne s'en rende compte. Pour ce faire, les auteurs utilisent un ensemble de règles pour gérer l'état d'une machine virtuelle, c'est-à-dire si celle-ci est sur-provisionnée (la tâche a trop de ressources par rapport à ce qu'elle consomme), ou sous-provisionnée (la tâche manque ou va manquer de ressources).

D'une manière générale, ce type d'approche est particulièrement adapté aux systèmes virtualisés. Dans de tels systèmes nous pouvons, comme expliqué précedemment, modifier la localisation des tâches, par le biais de la migration *live* (sans interruption de fonctionnement). Nous pouvons aussi modifier à la volée les ressources accordées aux machines virtuelles, c'est-à-dire modifier les parts de ressources de la machine qui sont attribuées à une machine virtuelle donnée. Ces techniques sont d'une manière générale celles observée dans l'approche cloud [102, 77].

Nous pourrons donc réagir, au vu de ces différents leviers, en fonction soit d'évènements précis, soit en fonction des différentes variation du système.

2.4.2 Autonomic/Frameworks

Etant donné que la réduction de la consommation d'énergie peut se faire à plusieurs niveaux, et doit se faire comme telle, il peut être préférable avoir une approche holistique en définissant des frameworks globaux. C'est le cas de Younge et al. dans [106], où les auteurs proposent justement un framework pour la gestion efficace de clouds afin de réduire la consommation d'énergie. Ce genre d'approche est nécessairement transversale puisqu'ils proposent une gestion hiérarchique partant du niveau le plus haut (*Green cloud*), et descendent jusqu'à l'ordonnancement *power-* et *thermal-aware*, en passant par la conception du data center et le controle des machines virtuelles du cloud. De cette manière, les économies d'énergies peuvent être atteintes en unifiant tous les niveaux de l'approche. Il faudra cependant bien rester dans certaines lignes de conduites, afin de toujours minimiser l'impact sur la performance, mais aussi minimiser l'impact inter-niveaux.

Une autre approche, décrite dans le framework GREEN-NET [36], consiste à fournir des outils pour la récupération des informations de consommation d'énergie et de les présenter aux utilisateurs, ce qui permet d'inclure ces utilisateurs dans cette décision de réduire la consommation, tout en mettant en place des mécanismes automatiques pour réduire la consommation d'énergie en respectant les besoins des utilisateurs.

Les approches autonomes s'inscrivent directement dans le réactif. En effet, même s'il peut y avoir des notions de planification dans le monde autonome, son fonctionnement représente bien cette réactivité au système, puisqu'il est basé sur une boucle comportant une partie de surveillance, une partie d'analyse des données de la surveillance, d'une partie de prise de décision en fonction de l'analyse, et d'une partie d'actions à effectuer, qui seront ensuite surveillée, et ainsi de suite. La surveillance du système sera donc toujours une étape nécessaire pour pouvoir effectuer des actions éclairées.

Ces approches autonomes sont souvent multi-niveau. Cela veut dire que chaque étape de la boucle autonome peut s'effectuer aussi bien au niveau composant qu'au niveau cluster. La centralité inhérente à ce genre d'approche amène cependant à faire des compromis en fonction des niveaux. Puisque nous sommes dans un contexte de système large échelle, nous aurons du mal à gérer à grain fin le niveau composant comme le niveau machine. Cependant, malgré ce problème, les approches autonomes réactives sont efficaces et permettent la gestion automatique du système sans intervention humaine. Khargharia et al. [64] adoptent une telle approche pour la gestion de la performance et de la puissance, pour optimiser la performance par watts tout en gardant la propriété de passage à l'échelle. Leur approche vise à réduire le gaspillage d'énergie en se

conformant à des contraintes de performances. Les auteurs utilisent ici une approche permettant à un gestionnaire autonome niveau cluster de résoudre un problème d'optimisation par contraintes afin de trouver les états optimaux de serveurs. Des expériences par simulations montrent que dans un système à plusieurs composants mémoire, des économies d'énergie de 72% en moyenne peuvent être atteintes avec leur approche dynamique par rapport à une approche statique. De plus, presque 70% d'économies supplémentaires peuvent être atteintes en utilisant une approche hiérarchique, c'est-à-dire une gestion à la fois au niveau du cluster et des serveurs.

Dans [44], les auteurs présentent Snooze, un gestionnaire autonome pour les économies d'énergie dans les clouds. Snooze implémente une approche multiniveau pour la gestion des machines virtuelles. Outre le placement initial de machines virtuelles, Snooze permet notamment de transférer des machines virtuelles d'un hôte vers un autre dans plusieurs cas. Le premier intervient lorsqu'un hôte est surchargé ou sous-chargé, auxquels cas on va distribuer la charge pour soit libérer l'hôte, soit le décharger un peu pour préserver la qualité de service. Le second sera périodique, ou le système tentera d'effectuer une consolidation complète pour permettre au gestionnaire de consommation d'énergie d'éteindre les hôtes inutilisés, après une période d'inactivité. Les auteurs montrent ensuite par expérimentation sur la plateforme Grid5000 avec 34 machines que leur système permet d'économiser jusqu'à 67% d'énergie sur des charges de travail réelles, comparé à une approche sans économies d'énergie.

Rodero et al. dans [93] présentent un gestionnaire autonome de cloud HPC permettant de gérer à plusieurs niveaux le compromis entre les économies d'énergie, la qualité de service et la gestion de température, afin de maximiser l'utilisation de ressources de l'infrastructure. Leur approche réactive utilise des leviers de migration de machines virtuelles, de DVFS ainsi que de fixation de puissance de CPU pour réagir aux différentes anomalies thermales. Les auteurs simulent ensuite leur infrastructure en exécutant charges basées sur des traces d'application HPC pour montrer que l'allocation de machines virtuelles telle que présentée permet d'augmenter l'efficacité énergétique de 12% et/ou d'optimiser la performance de l'application de 18% en fonction des buts d'optimisation définis. En termes de gestion réactive de l'énergie et de la température, utilisée en conjonction de l'allocation, permet de réduire la consommation d'énergie jusqu'à 9% et jusqu'à 12% de réduction de temps de calcul par rapport à une approche usuelle *temperature-aware*.

Le framework d'architecture pour clouds efficace en énergie proposé dans [21] introduit des notions de provisionnement de machines virtuelles, ainsi qu'une allocation et réallocation des machines virtuelles sur les machines physiques. Les auteurs utilisent la ressource qu'est le CPU pour consolider dynamiquement les machines virtuelles, et ainsi réduire de manière significative la consommation globale de l'infrastructure. Plus concrètement, les auteurs implémentent une politique de replacement des machines virtuelles basée sur un double seuil (bas et haut), ainsi qu'une politique de minimisation du nombre de migration, afin de limiter le nombre de migration, par exemple depuis un hôte surchargé. Le framework est ensuite évalué selon deux métriques : la consommation d'énergie totale de l'infrastructure, et le pourcentage de violations de SLA. Les résultats montrent une économie d'énergie allant jusqu'à 66% en comparaison à une approche sans migrations, en gardant un niveau de violations de SLA bas (1.1%). Cependant, les auteurs notent qu'en acceptant une dégradation de performance plus grande il est possible de réduire bien plus l'énergie consommée. Le choix revenant bien sûr aux fournisseurs de clouds, en fonctions des contrats de qualité de service négociés préalablement.

Par rapport aux autres approches, les frameworks et approches autonomes transversales se confronteront souvent aux problèmes décrits en section 2.2.5. Ces approches multi-niveaux auront des problèmes d'interaction entre les différents moyens d'actions lorsqu'utilisés de façon concurrente.

2.5 Synthèse

La gestion de tâches est un problème complexe, multi niveau, et qui nécessite une compréhension étendue des systèmes que l'on manipule. Si l'on met de côté les problèmes techniques tels que la mauvaise adéquation qu'il peut y avoir entre les besoins de réduire l'énergie, et le peu de moyens fournis pour le faire, il subsiste un bon nombre de problèmes complexes et intriqués.

Le parallèle peut être fait avec le monde de l'informatique mobile, où la gestion de l'énergie est aussi cruciale, et les techniques de réduction de celle-ci similaires. Il faudra cependant différentier le monde filaire avec celui du mobile dans la mesure où dans le mobile, nous aurons une contrainte "dure" sur la consommation d'énergie, autrement dit la batterie de l'appareil, alors que dans le monde du cloud, nous aurons une limite plus "douce", puisque cela sera un coût d'une énergie virtuellement infinie.

Il existe cependant bon nombre de techniques à tous les niveaux qui se révèlent performantes. Bien que des approches globales existent, à chaque niveau subsiste un manque de prise en compte de tous les freins que l'on peut rencontrer. Par exemple dans le cadre de la consolidation de machine virtuelles, les approches se révèlent prometteuses, mais ne prennent souvent pas en compte tous les problèmes liés à la migration de machines virtuelles. Il apparait dans ce cas un manque de modèles simples et précis pour modéliser cette migration et les problèmes associés. Un chemin considérable reste encore à faire pour une approche globale et précise à tous niveaux.

Comme nous avons pu le voir, il existe un nombre important de leviers que nous pouvons prendre en compte. Certains sont à un niveau composant alors que certains sont à un niveau d'infrastructure. Il convient donc de choisir les bons leviers pour arriver aux meilleures économies d'énergie, avec pas ou peu de perte en performance de tâches. Étant donné la complexité du problème lorsque l'on utilise un trop grand nombre de leviers, nous avons fait le choix de n'en utiliser que certains. Nous avons notamment choisi de ne pas utiliser le levier du DVFS au profit de l'extinction des hôtes pour deux raisons principales.

Premièrement, nous nous plaçons dans un contexte de tâches en tant que services, qui sont des tâches qui n'ont pas de complétion, afin de pouvoir déporter la gestion des requêtes aux clients exécutant les services. Ainsi, puisque les services n'ont pas de terminaison, le choix du couple voltage/fréquence peut être considéré comme déjà associé à l'hôte, et positionné sur le couple le plus efficace en terme de performance par watt. En effet, nous pourrons avec notre modèle définir chaque couple comme un hôte différent. Cela enlève le gain en énergie que l'on peut gagner en réglant le DVFS en fonction de la charge actuelle, mais cela nous enlève aussi les interactions négatives qu'il peut y avoir entre l'allocation d'une tâche sur un hôte et la fréquence à laquelle il est réglé.

Deuxièmement, le choix de privilégier l'allumage et l'extinction des hôtes comme levier principal d'action est amené par le simple fait que pour la majorité des machines actuelles, une portion d'au moins 50% de la consommation de la machine vient du simple fait que celle-ci soit allumée. Ainsi, le plus gros potentiel à priori pour les économies d'énergie vient de cette extinction.

Nous avons pu aussi voir qu'il existe deux catégories de travaux. Ceux qui tentent de réduire la consommation d'énergie sous contrainte de performance, à savoir sans impacter aucunement sur celle-ci, et ceux qui tentent de réduire la consommation d'énergie, en minimisant la perte de performance. Si le premier peut être souhaitable dans une majorité de cas, il n'en reste pas moins que d'autres cas existent où l'on pourra impacter sur la performance. Le deuxième cas aura cependant un potentiel de réduction d'énergie bien plus grand, comme on peut facilement l'imaginer, puisqu'en s'enlevant cette contrainte, les décisions d'allocation de tâches seront plus libres.

Ensuite, il faut prendre en compte le fait que nous nous placerons dans le monde du cloud, autrement dit dans un contexte de système distribué à large échelle. Ainsi, il sera d'importance

maximale d'avoir des algorithmes passant à l'échelle, pour pouvoir être exécutés dans des environnements comprenant plusieurs milliers de machines virtuelles et physiques. Comme nous avons pu le voir, certaines des approches calculent la gestion optimale en énergie. Si nous allons suivre cette approche dans une certaine mesure, à des fins de comparaisons essentiellement, nous développerons aussi dans la suite un nombre d'algorithmes basés sur des heuristiques permettant d'arriver à des solutions sous optimales, mais en un temps acceptable. Calculer l'optimal nous permettra aussi de pouvoir observer les différents effets que peuvent avoir les différents aspects du système sur la consommation d'énergie. Autrement dit, il faudra pouvoir savoir jusqu'où il est possible de réduire la consommation d'énergie.

Les différentes caractéristiques du système que l'on va prendre en compte sont notamment un aspect important qui va diriger la manière dont nous allons effectuer les allocations. Certains travaux prennent en compte simplement le CPU au niveau des hôtes, alors que d'autres prennent en compte aussi la mémoire, le disque ou la bande passante. Certains prennent aussi en compte l'infrastructure globale, notamment les équipements de refroidissement. Il apparait que tous ces aspects sont importants, et c'est pourquoi prendre en compte un nombre maximum est important. Nous prendrons pour notre part par la suite deux ressources au niveau d'un hôte (bien que notre approche soit facilement généralisable à n ressources), le CPU car c'est le plus important consommateur d'énergie dans les serveurs, mais aussi la mémoire, car il est important de prendre en compte une ressource de plus pour en observer les interactions, mais aussi car la mémoire peut facilement être un facteur limitant dans la performance des applications.

Ce faisant, il apparait dans les travaux présentés ci-avant une nécessité de prendre en compte des systèmes de cloud réels. À l'heure actuelle, il est commun d'avoir des clusters comportant un ensemble de machines différentes, que ce soit dû à une amélioration du cluster ou à différentes réparations au fil du temps. Ainsi, l'hétérogénéité a une place importante dans le paysage des clusters, et il est important à prendre en compte, puisque les efficacités énergétiques des différentes machines ne sera pas la même, impactant ainsi l'allocaton de tâches.

Un certain nombre des articles présentés précédemment prennent en compte le refroidissement, soit en tant que tel, soit en effectuant une allocation avec considération de température, soit en prenant en compte directement les équipements de refroidissement. Il est important, sinon nécessaire, de prendre en compte le refroidissement dans la gestion de tâches. En effet, le refroidissement est une part importante de la consommation globale d'un data center (entre 15% et 30%). De plus, l'allocation de tâches, du fait de la répartition de la charge, aura un effet direct sur les équipements de refroidissement. C'est pourquoi ce sera un paramètre que nous prendrons en compte par la suite.

Enfin, il apparaît qu'il est nécessaire de prendre en compte la dynamicité du système. La charge de travail n'est pas constante au fil du temps, et l'adaptibilité du système aux différents changements doit fait partie du processus de la gestion des tâches. Ainsi, la gestion de tâche repose en grande partie sur la capacité du système à pouvoir migrer les machines virtuelles d'un hôte sur l'autre. Cette migration est coûteuse en ressources, et il est nécessaire de prendre en compte le coût de migration dans le processus de décision. On évitera notamment d'avoir trop de migrations concurrentes, ce qui amènerait un surcoût trop important.

Au final, il est préférable de tendre vers une approche transversale, multi paramètres, en utilisant avec parcimonie divers leviers, pour atteindre le plus d'économies d'énergie possible. Nous tenterons donc par la suite d'effectuer une gestion de tâches prenant en compte différentes ressources de machines (CPU, RAM), l'hétérogénéité des machines et l'infrastructure de refroidissement, ainsi que les coûts à la fois en temps et en ressources des migrations de machines virtuelles. La table 2.2 présente les différents aspects de certaines des approches présentées précédemment.

Référence	Type de contrainte	Type de ressource	Optimal	Coût de migration	Refroidissement	Hétérogénéité
[77]	Performance	CPU	Oui	Nombre de migrations	Non	Non
[103]	Temporelle	CPU/MEM/IO	Oui	Non	Non	Non
[38]	SLA	Tous	Local	Coût du transfert réseau	Non	Oui
[64]	Performance	RAM	Oui	Non	Non	Oui
[22]	SLA	CPU	Non	Non	Non	Non
[25]	SLA	CPU/MEM	Non	Temps et ressources	Non	Non
[31]	Temps de réponse	CPU	Non	Nombre de Migration	Oui	Non
[87]	Performance	CPU	Oui	Fixe en énergie	Non	Oui
[92]	Temps d'exécution	CPU/MEM/IO	Non	Non	Non	Non
[23]	Performance,Énergie	CPU/MEM	Oui	Non	Oui	Oui
[43]	Énergie	CPU	Non	Non	Non	Non
[60]	Performance et Énergie	CPU/MEM	Non	Non	Non	Non
[99]	Performance et Énergie	CPU/Disque	Oui	Non	Non	Oui
[96]	Temps d'exécution, Energie	CPU	Non	Non	Non	Oui
[46]	SLA	CPU	Non	Non	Non	Non

TABLE 2.2 – Synthèse des approches

Chapitre 3

L'allocation de services

L'allocation de tâches peut prendre, comme décrit précédemment, plusieurs formes, et répondre à des problèmes bien différents. Dans ce chapitre, nous décrirons l'allocation de tâches telle qu'elle va être utilisée dans notre cadre.

3.1 Quel est le système ?

Dans le cadre de nos travaux, nous nous sommes placés dans un contexte de clouds. C'est-à-dire un contexte, où nous possédons une infrastructure virtualisée, et les tâches sont contenues à l'intérieur de machines virtuelles. Chaque tâche étant hébergée par une machine virtuelle, nous avons donc la capacité au niveau de l'utilisateur, de demander un pourcentage pour chaque ressource de la machine hôte. De plus, une infrastructure virtualisée implique aussi une possible migration des machines virtuelles d'une machine physique vers une autre. Au niveau de l'infrastructure elle même, nous prendrons un cloud, constitué de plusieurs clusters géographiquement distribués, un cas qui arrive quand le fournisseur de services, pour offrir une meilleure qualité d'expérience à ses utilisateurs, va construire des data centers au plus près de ses clients. Par exemple, on peut avoir un cloud possédant en Europe plusieurs centre de données dans différents pays. C'est le cas de Google, qui possède en Europe trois data centers : un à Hamina en Finlande, un à St. Ghislain en Belgique et un dernier à Dublin en Irlande [51]. C'est aussi le cas des emplacements périphériques des centres d'Amazon pour le service de distribution de contenu CloudFront, qui possède des data centers à Amsterdam, Dublin, Francfort, Londres, Madrid, Milan, Paris, Stockholm[11].

Ces data centers étant distribués, potentiellement construits à des moments différents, n'auront pas tous été dotés des mêmes machines, ni de la même infrastucture de refroidissement. A l'intérieur d'un data center même, nous pourrons avoir des machines différentes, dans le sens où par exemple une augmentation de capacité aura été effectuée, et ainsi le cluster possèdera des serveurs moins récents que d'autres.

Par conséquent, nous avons une infrastructure de clusters possédant des hôtes homogènes à l'intérieur du cluster, mais des clusters hétérogènes entre eux. Ce qui veut dire que nous aurons des capacités de serveurs différentes sur chaque cluster, mais ce ne sera pas le cas à l'intérieur du cluster.

Au niveau de la consommation d'énergie inter clusters, nous aurons une consommation différente des hôtes, dû aux spécifications matérielles différentes, mais aussi à l'intérieur d'un même cluster, au sein d'un même rack, nous aurons des consommations différentes. Cet effet à été observé sur grid5000 dans [35], et peut être tout simplement attribué à la chaleur montante

des serveurs situés en bas du rack, induisant une plus grosse chaleur (et donc consommation) des serveurs situés en hauteur. En effet, une chaleur plus importante, qu'elle soit induite par la charge du processeur, ou par l'environnement, induira potentiellement une augmentation de la vitesse du ventilateur, ce qui amènera une consommation plus importante. De plus, il existe une variation de consommation de puissance entre les différents processeurs d'un même modèle comme montré dans [14], où les auteurs étudient la variabilité de consommation de puissance entre différents processeurs Intel Core i5-540M. Leur données montrent une variabilité allant de 7% à 17% en fonction des différentes applications et des différentes configurations de DVFS des processeurs.

L'infrastructure de refroidissement des différents data centers est aussi à prendre en compte, celle-ci consommant une grosse partie de la consommation totale du data center. Ici, l'infrastructure de refroidissement sur laquelle nous allons agir est celle qui conditionne et gère les flux d'air dans les data centers. Nous supposerons donc par la suite que le refroidissement du cluster se fait par acheminement d'air froid, ce qui n'est pas le cas dans tous les centre de données. Ainsi, dans un cluster, nous aurons une consommation d'énergie totale dépendant à la fois de la consommation des serveurs, et de l'infrastructure de refroidissement. Cette infrastructure de refroidissement, est appelée CRAH/CRAC pour *Computer Room Air Handler/Conditionner*, le CRAH étant ce qui amène l'air froid sur les serveurs, et le CRAC étant ce qui refroidit effectivement ce qui doit être acheminé. Celle-ci sera dépendante de l'utilisation du cluster. En effet, de la même manière que pour les serveurs eux-mêmes, une grande utilisation des serveurs amènera une hausse de température, ce qui aura pour effet d'augmenter la vitesse des ventilateurs pour le refroidissement afin de maintenir une température basse constante, ce qui amènera à son tour une plus grande consommation d'énergie.

Nous supposerons que nous pourrons agir sur ces CRAC/CRAH, afin de pouvoir les allumer ou les éteindre en fonction des besoins. Si le groupe de serveurs directement refroidis par un CRAC/CRAH n'est pas utilisé, donc éteint, nous éteindrons aussi l'infrastructure de refroidissement, afin d'avoir une économie d'énergie substantielle. Nous ne prendrons pas en compte la perturbation des flux d'airs, ce qui est cohérent dans un cadre de certains data centers modulaires par exemple. Ceux-ci peuvent être de type conteneurs, où chaque unité, chaque conteneur, contient à la fois les serveurs, le refroidissement et l'alimentation de manière autosuffisante. Cette supposition peut être aussi cohérente si l'on suppose que l'on est en mesure d'accepter une hausse dans la température globale de la salle, dûe à la perturbation des flux d'air. Ainsi, lorsque l'on décidera d'éteindre les CRAC/CRAH, nous pourrions voir une augmentation de la température moyenne de la salle, mais restant toujours en dessous des températures recommandées par l'ASHRAE (American Society of Heating, Refrigerating and Air-Conditioning Engineers) [5]. Les data centers ont plus souvent tendance à être trop refroidis, et les degrés en trop sont les plus chers. La relation entre la puissance et la température n'étant pas linéaire [45], une augmentation faible de température induira une faible augmentation de consommation de puissance. Par contre, une augmentation forte de température induira une augmentation proportionnellement plus forte de consommation. À l'inverse, réduire la température de peu de degrés réduira grandement la consommation d'énergie, et les degrés suivants réduiront en proportion moins cette consommation.

Pour plus de clarté, et pour résumer toutes ces suppositions, prenons un exemple montrant dans quel cadre nous nous trouvons, représenté en figure 3.1.

Nous avons un fournisseur de services clouds distribuant des services à des utilisateurs en Europe. Ainsi, ce fournisseur a décidé d'implanter ses serveurs à différents endroits, afin de pouvoir mieux servir les différentes zones géographiques. Cette infrastructure comprend 3 clusters géographiquement distribués, situés en France, Royaume Uni et Norvège. Pour chaque cluster, nous avons des spécifications différentes pour tous les hôtes. Ici, le cluster du Royaume-Uni

possède des hôtes avec chacun un processeur de 2.2 GHz et une mémoire RAM de 8 GB, alors que le cluster de France possède des hôtes avec un processeur de 2.6GHz. Bien que tous les hôtes d'un même cluster soient identiques au niveau matériel, ils consomment une puissance maximale différente (par exemple allant de 150W à 250W pour le cluster de Norvège).

Pour finir, nous avons les systèmes de refroidissement qui sont utilisés pour refroidir uniquement un ensemble défini d'hôtes. Ici pour l'exemple nous avons deux CRAC/CRAH par clusters, chacuns répartis pour refroidir la moitié des hôtes des clusters. Il faudra noter cependant que les CRAC/CRAH sont utilisés dans les infrastructures réelles sur bien plus de serveurs.

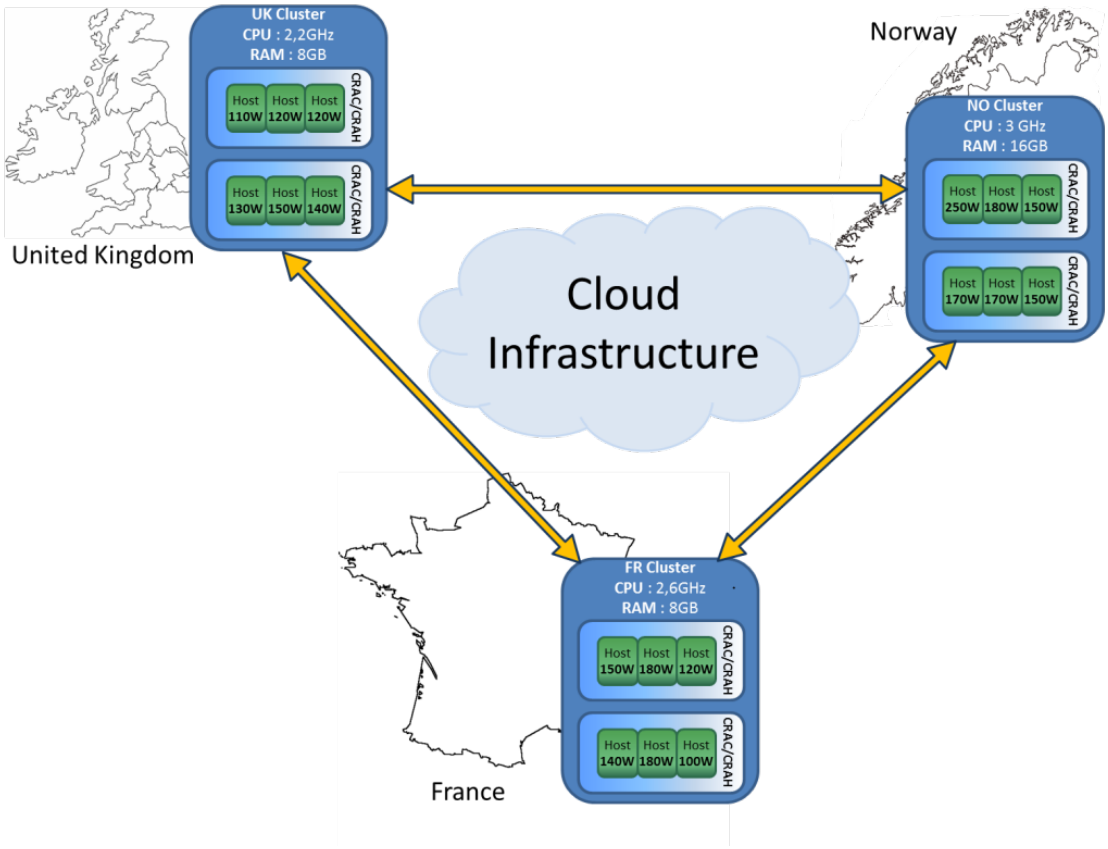


FIGURE 3.1 – Un exemple de cloud

3.2 Quel est le problème ?

3.2.1 Consommation d'énergie

Si l'on prend exactement la petite infrastructure décrite dans l'exemple en figure 3.1, les problèmes de consommation d'énergie n'apparaissent pas forcément. Dans les infrastructures de cloud réelles, nous avons plusieurs milliers de serveurs à gérer. Cela implique d'une part une consommation d'énergie très importante, lorsque l'on voit par exemple que le second superordinateur le plus rapide du monde en juin 2012 consomme plus de 12MW [98]. D'autre part, l'impact

environnemental n'en est que plus grand, autant au niveau du CO2 émit que de l'impact possible sur l'environnement de certaines techniques de refroidissement, comme la consommation d'eau induite par le refroidissement à eau. Enfin, plus de machines induisent plus de déchets, puisque les machines ne sont pas totalement recyclables, et ainsi, il est important de pouvoir prendre en compte le cycle de vie entier des data centers [16].

Le passage à l'échelle induit donc des coûts importants si aucune action n'est entreprise pour contrer ces effets néfastes. Il y a cependant des effets positifs à considérer lorsque l'on va économiser de l'énergie. Le premier et probablement le plus important est le gain financier. En effet, une étude a montré que les l'argent dépensé dans l'achat des serveurs des data centers était en 2008 aux États-Unis comparable à l'argent dépensé pour le refroidissement et l'électricité. De plus, la tendance indique que le coût en électricité et refroidissement dépassera le coût d'achat des data centers dans les années suivantes. Ainsi, le potentiel de réduction de la facture d'électricité via réduction de la consommation d'énergie est important [89].

Un autre effet positif à considérer est l'impact moindre du surdimensionnement des data centers au niveau coût d'exploitation. En effet, la plupart des clusters sont dimensionnés de manière à pouvoir soutenir des pics de charge, et ce surdimensionnement coûte cher en fonctionnement. Ainsi, si l'on implémente des techniques permettant de réduire le nombre de serveurs allumés en fonction de la charge globale de l'infrastructure, le data center devrait se trouver en théorie toujours bien dimensionné.

Économiser de l'énergie est donc à la fois une nécessité écologique et un attrait financier pour les entreprises, et bien que divers moyens existent pour réduire la consommation d'énergie des clusters, il faudra à chaque fois prendre en compte les différents impacts potentiels (positifs ou négatifs) induits par la consommation, et par la réduction de cette consommation.

3.2.2 Qualité de service

Les fournisseurs de services cloud doivent par contrat fournir un service de qualité. Si ce n'est pas le cas, le fournisseur de service pourrait se voir pénalisé financièrement par le client avec qui il avait passé un contrat, si celui-ci n'était pas respecté. Par exemple, un client pourrait négocier par contrat un temps de réponse du service inférieur à 500ms 95% du temps. Si le fournisseur n'était pas capable d'effectivement avoir un temps de réponse de la sorte, et était à moins de 500ms seulement 90% du temps, il se verrait obligé par le contrat de qualité de service (via Service Level Agreement ou SLA en anglais), de payer une pénalité.

Les pénalités de non respect de la qualité de service peuvent ne pas être directes, si par exemple l'insatisfaction des clients implique une mauvaise réputation de l'entreprise et une perte de clients potentiels.

Le respect de la qualité est donc un problème très important aux yeux à la fois des clients mais aussi des fournisseurs, et le non respect de celle-ci peut avoir des conséquences néfastes diverses pour le fournisseur de service.

Cette qualité de service se traduit souvent au niveau du fournisseur de service en termes de performances. Ainsi, pour garantir un temps de réponse acceptable pour les services web hébergés dans un cloud, il faudra que celui-ci ne soit pas surchargé, et par conséquent ne pas arriver dans un système où une requête arrivant doive rester en attente en raison d'un trop grand nombre de requêtes. Pour ce faire, il faut que le data center soit bien dimensionné, et il faut que la puissance de ressource (comme le CPU ou la bande passante) disponible soit suffisante.

Ces ressources disponibles auront cependant des effets différents à considérer. Il y aura souvent une différence entre un manque de puissance processeur, et un manque de mémoire. Un manque de puissance CPU entrainera un ralentissement du programme qui s'y exécute, alors qu'un manque de mémoire pourra potentiellement entrainer un dysfonctionnement total du programme.

Toutefois, comme expliqué précédemment, les ressources mises à disposition de l'utilisateur ont un coût en puissance électrique et par conséquent financier. Augmenter la performance des infrastructures coûte de plus en plus cher, et il est plus difficile d'effectuer des économies d'énergie.

À l'image de l'augmentation de la vitesse des processeurs, qui a longtemps été dirigée par la "loi de Moore" et s'est vue heurter une barrière dans la miniaturisation, nous observons un phénomène similaire dans le monde des data centers, et en particulier dans le monde du calcul haute performance. Il apparait que l'augmentation de la puissance entraine une augmentation de la consommation d'énergie, ce qui mène à une barrière, celle où une centrale électrique serait nécessaire pour pouvoir faire fonctionner son data center. Ceci implique, toujours dans le monde du HPC notamment, que chaque amélioration de performance induit un coût de plus en plus grand [67].

La qualité de service rendue aux utilisateurs restera donc la préoccupation première des fournisseurs de services, mais il faut bien prendre en compte le coût de cette qualité de service, autant dans sa globalité qu'en prenant en compte des paramètres plus spécifiques comme l'impact des ressources sur la qualité de service. Il faudra pour être plus efficace être capable d'accepter une dégradation faible de la qualité de service pour permettre des gains plus grands dans d'autres domaines, comme dans le domaine de la consommation d'énergie comme nous le verrons par la suite.

3.2.3 Energie & Qualité de Service

Comme nous avons pu le voir, les économies d'énergies et la qualité de service sont deux points importants que nous devons prendre en compte, avec leur profits et leur désavantages.

Le problème est que ces deux buts bien distincts apparaissent antagonistes lorsque l'on veut les atteindre de façon concurrente. Il existe des relations directes et indirectes entre ceux-ci. C'est pourquoi lorsque l'on essaye de réduire la consommation d'énergie d'un système, cela amène souvent à une baisse de performance, et de la même manière lorsque l'on veut avoir une qualité de service parfaite, il faudra compter sur une consommation d'énergie augmentée.

Comme il faudra dans tous les cas faire des choix, on peut être tenté de réduire la consommation d'énergie et d'augmenter la qualité de service de manière concurrente, en tentant d'atteindre un bon compromis. Pourtant, il apparait difficile de décider quel sera le meilleur compromis.

Prenons comme exemple une allocation de tâches où l'on est capable d'atteindre une réduction de 1kW de la puissance consommée tout en perdant 8% de qualité de service, et une allocation où l'on atteint 4% de réduction de qualité de service tout en réduisant la consommation de puissance de 500W. Comment dire quelle allocation devra être préférée ? C'est tout le problème de l'optimisation multi-objectifs, lorsque ces objectifs sont à la fois antagonistes et différents en poids. Il existe pourtant une allocation qui sera plus performante que l'autre dans sa globalité, mais celle-ci sera dépendante du contexte dans lequel on se trouvera. Si l'on poursuit notre exemple, on peut facilement imaginer un cloud de services web où une perte de performance de 8% sera acceptable dans les creux de charge, ce qui fera que la première situation sera la meilleure. On peut aussi facilement imaginer un cloud faisant tourner des tâches plus orientées performance et calcul, où la première situation ne sera pas acceptable et nous préfèrerons la deuxième.

Il convient aussi de parler des possibles paliers existant dans ce genre d'optimisation. Cela veut dire que lorsque l'on tente de réduire l'énergie, il apparaitra qu'éteindre des CRAC/CRAH ou à plus petite échelle éteindre des machines fera apparaître des paliers d'énergie modifiant la performance du système.

Prenons une nouvelle fois un exemple. Si notre allocation est orientée vers la consommation d'énergie, on peut tenter d'éteindre complètement un sous ensemble de machines avec le

CRAC/CRAH associé. Cependant, si la demande est trop importante et que l'on ne parvient pas à réduire la charge suffisamment pour éteindre ces infrastructures, nous verrons une réduction dans la performance qui ne servira que peu à réduire la consommation d'énergie. Cela est dû au fait qu'une partie importante de la consommation des machines vient seulement du fait d'être allumée. Par conséquent, si l'on réduit la performance des tâches sans pouvoir éteindre CRAC/CRAH, l'allocation résultante sera potentiellement inefficace.

À l'inverse, si l'on parvient à réduire la performance de manière à éteindre ce sous ensemble avec son système de refroidissement, nous gagnerons énormément en énergie. Ce qui voudra dire que les derniers paliers de réduction de la performance auront été très efficaces en terme de réduction d'énergie, et que le compromis sera intéressant.

Tous ces points ajoutent à la difficulté de pouvoir évaluer la performance d'une allocation, et de décider de métriques pour cette évaluation. Il conviendra toutefois de prendre en compte le contexte du problème dans la résolution. En effet, si nous avons un fournisseur de services qui se trouve dans une contrainte forte en consommation d'électricité, dû par exemple au budget donné par le fournisseur d'électricité, il faudra pouvoir optimiser la performance tout en satisfaisant cette contrainte. L'opposé avec une contrainte forte en QoS est aussi possible. Pourtant, si nous n'avons pas de contraintes, il faudra se conformer aux désirs soit de l'utilisateur soit du fournisseur pour définir un compromis valable, et ainsi favoriser soit l'énergie soit la performance.

3.3 Hypothèses

Le problème d'optimisation, comme décrit précédemment, est complexe, vaste et comporte maintes interactions rendant le problème global extrêmement difficile à résoudre dans les systèmes large échelle. Trop d'interactions impliquent une masse de calculs trop importante et trop complexe lorsque l'on veut faire de la gestion optimale.

Il faudra donc réduire le problème global à un sous problème. Pour ce faire, nous avons posé différentes hypothèses, qui sont soit simplificatrices, c'est-à-dire que l'on modifie un aspect du problème d'allocation pour s'enlever un élément de complexité, mais qui peuvent être aussi des hypothèses permettant de réduire ou de préciser le cadre et l'étendue de la solution que nous allons proposer.

3.3.1 Type de tâches

La première hypothèse que nous serons amenés à faire concerne les tâches auxquelles nous allons nous intéresser. Premièrement, il faudra définir quel type de tâches nous allons considérer. Chaque tâche sera exécutée à l'intérieur d'une machine virtuelle. Nous pourrons donc assimiler la tâche à sa machine virtuelle, sachant que les tâches ne seront pas interdépendantes. De la même manière, nous ignorerons le surcoût induit par la virtualisation, en l'assimilant à la consommation de ressources de la tâche.

3.3.2 Services "constants"

Dans nos travaux, nous prendrons en compte majoritairement des tâches qui n'impliquent que peu de variation de leur besoins en ressources. Concrètement, cela veut dire que sans aller jusqu'à définir les tâches comme constantes, nous pourrons les définir variant peu ou pas dans le temps.

Cette hypothèse implique qu'au niveau du monitoring du système, les mesures seront précises à un instant t, et surtout seront valables pendant un certain temps, dans le sens où la variation

pendant ce temps sera mineure. Ce qui nous permettra aussi de pouvoir dire que les actions que nous entreprendrons et qui prennent du temps, prendront effet dans un système encore valable.

Si l'on prend par exemple une tâche qui consomme 50% du CPU de son hôte, nous prenons une décision sur cette base de 50% de consommation, et décidons de migrer cette tâche depuis son hôte vers un autre, il faudra que quand la migration se termine, la tâche soit toujours avec une consommation autour de 50%, si les deux machines ont la même capacité de calcul. Par conséquent, si cette tâche avait pendant la migration changé fortement de besoins en consommation CPU, pour plus ou moins, l'allocation résultante n'aurait pas forcément été pertinente.

Cet effet est plus problématique lorsque les besoins augmentent pendant la migration que lorsqu'ils baissent, car cela peut générer une baisse de qualité de service dans le cas d'une hausse, ce qui n'est pas le cas lors d'une baisse. En effet, si une tâche demande plus de ressource pendant la migration, cela va non seulement impacter sa propre allocation, mais aussi la capacité des autre tâches à demander plus de ressources. Ainsi, une hausse des besoins impactera les autres tâches de manière négative, ce qui n'est pas le cas lorsque les besoins d'une tâche baissent.

3.3.3 Services infinis

Nous nous plaçons dans un contexte comprenant des tâches de type "services". C'est-à-dire des tâches qui fournissent un service à des utilisateurs, ce qui est souvent le cas dans les services de type clouds. Par exemple, une ensemble de machines virtuelles peut contenir une infrastructure de type service web, base de données, etc... Ainsi, la qualité de service sera modélisée dans ce cas précis par le temps de réponse donné aux utilisateurs.

Ces services sont considérés comme "infinis". Bien que cela n'ait pas vraiment de signification dans le monde réel, il apparait que ce type de tâches infinies soit en fait les services qui n'ont pas de terminaison prévue.

Contrairement aux tâches de type calcul haute performance qui se terminent le plus vite possible, les services n'ont pas de terminaison, et sont là pour fournir un service aux usagers. Cependant, nous pourrons aussi inclure des tâches non infinies, mais suffisamment longues en exécution pour permettre une action à la même échelle que les services sans terminaison. Cette échelle de temps peut être de l'ordre de l'heure ou de la journée, en fonction des besoins. Il faudra donc que ces tâches aient une échelle de temps d'exécution bien plus grande que l'échelle du temps d'action de la gestion. Par exemple des tâches d'une durée de l'ordre de plusieurs heures, alors que les actions auront une durée de l'ordre de plusieurs minutes.

Le fait de ne pas avoir de tâches finies implique aussi un changement au niveau du placement lui même. Nous ne pourrons ainsi pas faire de scheduling ni d'allocation contrainte dans le temps, car les tâches ne se finissent pas, et nous ne pouvons donc pas les arrêter ni les mettre en queue, puisque le service doit être assuré en tout temps.

De ceci découle aussi les métriques qui seront différentes que pour les tâches de type calcul haute performance, dans la mesure où la métrique de performance n'est plus le temps de fin d'exécution de la tâche mais le temps de réponse pour l'utilisateur. Le temps de réponse présente pourtant l'avantage d'être plus permissif à un instant donné (au niveau qualité de service), puisqu'une baisse soudaine de performance n'aura pas le même effet. Par exemple, nous pouvons définir que le temps de réponse d'un service est acceptable tant qu'il reste en dessous de 3 secondes. Ainsi, une baisse de performance faisant passer d'un temps de réponse qui double de 0.5 seconde à 1 seconde ne sera pas un mauvais résultat.

3.3.4 Hétérogénéité

Les clouds sont souvent hébergés dans un ensemble de clusters, souvent géographiquement distribués (cf : figure 3.1). Par conséquent, il est difficile que l'infrastructure globale soit parfaitement homogène. Cela est aussi vrai à l'intérieur d'un même cluster, lorsqu'une augmentation matérielle est effectuée, nous aurons souvent deux types de machines différentes. Par conséquent, dans notre étude, nous prendrons en compte l'hétérogénéité inter cluster du cloud. C'est-à-dire un ensemble de cluster homogènes, hétérogènes entre eux.

Cela impliquera à notre niveau qu'une application exécutée sur un cluster, ne prendra pas forcément le même pourcentage de ressource en fonction des hôtes. Cela implique aussi la prise en compte donc au niveau hôte (et donc cluster) de l'efficacité energétique des machines. Des nouveaux problèmes émergeront donc de cette prise en compte, dans la mesure où prendre en compte l'hétérogénéité des clusters nécessite d'abord une connaissance à priori des différents clusters, mais nécessite aussi de prendre en compte l'infrastructure dans sa globalité.

En effet, si l'on avait seulement des serveurs homogènes, la prise en compte d'autres paramètres comme les infrastructures de refroidissement permettrait un classement direct des différentes machines en terme d'efficacité énergétique, mais cela devient plus difficile quand on doit prendre en compte à la fois l'hétérogénéité des machines et des CRAC/CRAH.

Au final, nous aurons un modèle comprenant des clusters homogènes, qui sont hétérogènes entre eux. Ce qui peut trivialement être étendu à une hétérogénéité totale, si l'on s'affranchissait du paradigme cloud, pour réduire la taille du cluster à un serveur unique. On peut donc se poser la question de quels sont les avantages d'une telle hypothèse ? Le fait d'avoir des clusters homogènes est d'une part plus cohérent avec l'approche cloud, et donc avec le monde réel, et d'autre part comme nous le verrons par la suite amènera une allocation plus performante, du fait notamment des infrastructures de refroidissement.

3.3.5 Action sur l'infrastructure de refroidissement

La prise en compte au niveau gestion d'infrastructures types cluster des différentes infrastructures de refroidissement n'est pas traditionellement pris en compte par les approches usuelles. Il est souvent difficile dans des clusters actuels d'avoir la possibilité d'agir sur celle-ci. Par exemple, les CRACs ne sont pas forcément dotés de technologies permettant le contrôle de la variation de la vitesse de la circulation de l'air. Bien que certaines approches aient implémenté ces technologies, celles-ci se cantonnent souvent à des algorithmes de gestion simple au niveau cluster, comme dans [20] où la gestion des ventilateurs réagit à la charge globale des machines du cluster.

Les infrastructures de refroidissement étant grosses consommatrices d'électricité, il est intéressant de pouvoir agir dessus. C'est pourquoi nous supposerons que nous pouvons agir sur l'infrastructure de refroidissement des clusters, afin d'éteindre celles-ci pour économiser de l'énergie.

Nous prendrons une approche similaire à [57] où les auteurs implémentent un algorithme de gestion de charge simple visant à éteindre une partie du cluster. Leur modèle de refroidissement comprend comme le notre un ensemble de CRAC/CRAH liés à un ensemble d'hôtes.

Plus précisément, nous prendrons en compte des clusters de type modulaires (de type conteneurs par exemple) [61] où nous avons une autosuffisance au niveau d'un sous ensemble de machines du cluster par rapport à son espace, son refroidissement, et son alimentation. Nous nous plaçons aussi plutôt dans un contexte où nous avons des CRAH dissociés des CRAC avec ventilateurs variables, ou alors des CRAC/CRAH que nous pouvons éteindre. Soit nous pourrons éteindre le refroidissement des machines, si leur état nous le permet, soit nous pourrons réduire la vitesse des ventilateurs au minimum, en assimilant le coût de fonctionnement de base du refroidissement (i.e : hors surcoût lié à la vitesse des ventilateurs) aux coûts de base du cluster,

c'est-à-dire les coûts sur lesquels nous ne pourrons agir (comme l'alimentation ou l'éclairage).

Au final, cette hypothèse va restreindre l'éventail de type de clusters que nous étudierons. Par exemple nous ne pourrons pas prendre en compte des clusters nécessitant un flux d'air bien précis, qu'il ne faudra pas interrompre sous peine d'empêcher le bon refroidissement de l'infrastructure.

Il faudra souligner le fait que l'allocation de tâches énergétiquement efficace agit aussi indirectement sur le refroidissement du cluster. Ainsi, si l'on alloue des tâches de manière intelligente, de manière à consommer moins d'énergie au niveau des hôtes, indirectement nous aurons aussi moins de chaleur générée, et par conséquent moins de besoins en refroidissement.

Il faudra aussi regarder le modèle de la consommation d'énergie, puisque les effets de la chaleur et de la consommation d'énergie sont d'une part non linéaires, et d'autre part reliés entre les différentes parties d'un cluster. Nous verrons par la suite dans la section 4.2.2 quel modèle nous avons choisi, et pourquoi. Il faut en effet choisir un modèle suffisemment simple et pas trop éloigné de la réalité, afin de prendre des décisions cohérentes. Un modèle simple nous permettra d'une part à donner une bonne compréhension à ceux qui s'en servent, et d'autre part à former une base simple à partir de laquelle nous pourrons étendre le modèle.

3.3.6 Reconfiguration de Machines Virtuelles

Nous nous plaçons dans une approche de type cloud computing, ce qui veut dire que nous possédons une infrastructure virtualisée. Comme mentionné précédemment, un des avantages de la virtualisation est la possibilité de réduire ou d'augmenter à la volée ce qui est effectivement alloué à une tâche (ou plutôt à la machine virtuelle qui la contient) en terme de ressources machine.

Ainsi, nous avons la capacité de reconfigurer les machines virtuelles en prenant des décisions par rapport à ce que la tâche consomme sur le moment. Cela implique que nous pouvons implémenter des techniques de prédiction de la consommation actuelle des machines virtuelles, et ainsi maximiser l'utilisation des ressources par les machines virtuelles.

Les tâches sont ainsi définies comme adaptables, c'est-à-dire qu'une fois que les contrats de qualité de services ont été négociés, nous pouvons toujours bouger la quantité de ressources allouées aux tâches, et si la consommation effective de la tâche n'excède pas la quantité effectivement allouée, il n'y aura pas de violation de SLA.

Cela implique aussi une connaissance à priori des clients pour faire une demande précise concernant les tâches qu'ils ont à exécuter, ce qui peut se révéler problématique dans certains cas.

3.3.7 Migration de Machines Virtuelles

Dans la même lignée que la reconfiguration de machines virtuelles, et aussi permis par le contexte de cloud computing, nous avons la capacité de migrer des machines virtuelles, c'est-à-dire changer l'exécution d'une machine virtuelle d'un hôte sur un autre. Cette migration est souvent l'approche de choix pour les services de type cloud, puisque celle-ci va permettre une consolidation des machines virtuelles sur les différents hôtes afin d'économiser de l'énergie, et de maximiser l'utilisation des ressources. Autrement dit, nous permettre de regrouper les machines virtuelles sur un sous ensemble réduit de machines, afin de pouvoir éteindre les hôtes inutilisés. De la même manière, nous pourrons faire de l'équilibrage de charge pour limiter les violations de qualité de service liées à la sur-utilisation de l'infrastructure.

Ainsi, bien que nous puissions gérer plus efficacement notre infrastructure, cette technique sera à double tranchant. Si les résultats de la migration peuvent être très importants, une mauvaise

gestion peut aussi les rendre catastrophiques. C'est pourquoi il faudra prendre en compte le coût intrinsèque à la migration.

Dans nos travaux nous prendrons en compte la migration de machines virtuelles, puisqu'une grosse partie des économies en dépendent, mais nous prendrons aussi en compte la gestion de la migration, ainsi que son impact. On peut facilement penser qu'une machine virtuelle qui migre d'un hôte vers un autre sera consommatrice de ressources à la fois sur l'hôte source et sur l'hôte destination, et ainsi consommera plus d'électricité que la normale pendant la durée de la migration.

Nous supposerons enfin concernant la migration que l'infrastructure hébergeant les clusters virtuels que nous avons un système de stockage centralisé, comme un NAS (Network Attached Storage) local au cluster pour héberger les images des différents systèmes à l'intérieur des machines virtuelles. Cela permet au niveau de la migration au sein d'un cluster de n'avoir à transférer d'une machine sur l'autre que la mémoire locale des machines virtuelles, et ainsi grandement réduire le temps de migration. De plus, la migration est supposée être ce que l'on appelle *live*, ce qui veut dire que la migration est faite à la volée, sans interruption de service, juste avec une mineure interruption au niveau réseau, par opposition à une migration *offline* qui se fait en 3 phases : interruption, copie, redémarrage. Enfin, la migration sera supposée être intracluster. Bien que celle-ci soit possible inter cluster, elle posera le problème du transfert de données vers l'extérieur du cluster, qui se font sur des liens bien moins rapides que les liens entre serveurs d'un même cluster. Une migration inter cluster demanderait aussi de migrer non seulement la mémoire de travail d'une machine virtuelle, mais aussi l'image du système, puisque l'accès à un NAS depuis un autre cluster deviendrait trop long et dégraderait trop les performances de la machine virtuelle.

3.3.8 Extinction/Allumage des hôtes

Une des plus importantes hypothèse concerne la capacité que nous avons d'allumer et d'éteindre les machines à distance. Ce genre de technique est en train de se démocratiser dans les clusters les plus récents. Comme mentionné précédemment, le levier de l'allumage et de l'extinction des hôtes n'est pas sans comporter ses revers. Il faut en effet prendre en compte les dommages possibles qu'un trop fréquent redémarrage des hôtes peut apporter aux matériels, et par conséquent bien faire attention à ne pas trop fréquemment changer l'état des machines.

Nous pouvons cependant intégrer dans nos problèmes cet effet, pour le limiter.

Pour économiser une grande quantité de puissance, nous pouvons éteindre les machines inutilisées. Cela implique une relation proche entre la consolidation des tâches (et par conséquent la migration des machines virtuelles), et l'extinction des hôtes. Il faudra être proactif pour forcer l'extinction de certains hôtes, en forcant une reconfiguration du système pour avoir une exécution sur un ensemble plus faible de machines.

Enfin, il faudra prendre en compte le temps que peut prendre l'allumage et l'extinction des machines. Lorsque l'on allume une machine, cela prend du temps, pour lancer tous les services nécessaire au bon fonctionnement du système, mais aussi pour faire des vérifications d'intégrité sur le matériel, comme des vérifications d'intégrité de la RAM par exemple. Dans les clusters, le redémarrage des machines est de l'ordre de plusieurs minutes. Il faut noter qu'il existe des techniques d'extinction partielle des machines, similaire à la mise en veille. C'est le cas des techniques de suspend-to-RAM où l'état courant du système est stocké dans la RAM et la majorité des composant de la machine est éteinte. Par contre, bien que ce genre d'état puisse permettre des allumages et extinctions des machines en des temps de l'ordre de la seconde, ces états consomment de l'énergie dans la mesure où certaines parties de la machine restent allumées.

Ainsi, avec le temps de démarrage et d'extinction, le problème des trop fréquents redémarrages

devient plus important. Ainsi, non seulement il faut faire attention à ne pas trop allumer ou éteindre les hôtes, puisque ceux-ci consomment de l'énergie alors qu'ils sont inutilisables par les tâches pendant ce temps là, mais aussi car tenter d'éteindre des machines trop tôt peut amener des situations où la puissance de calcul restante n'est plus suffisante pour garantir un certain niveau de qualité de service. Par exemple prenons 2 tâches qui consomment 40% chacune, ainsi que 2 hôtes. Nous prenons la décision d'éteindre un des deux hôtes en consolidant les 2 tâches sur la même machine physique. Quand nous commençons à éteindre l'hôte resté vide, nous voyons qu'il y a une augmentation de la quantité de resssources consommée par les tâches jusqu'à 50%. Dans ce cas précis, nous voyons bien que la machine restée allumée est utilisée à sa capacité maximale, mais que la machine éteinte n'est pas encore prête à être rallumée, empêchant ainsi les tâches d'augmenter plus leur ressources utilisée, ce qui amènera des problèmes en terme de qualité de service. Pour palier à ce problème, il conviendra de ne pas trop consolider aggressivement, afin de laisser de la marge de manoeuvre pour l'augmentation possible des ressources.

Le même problème existe pour l'allumage trop tôt des machines, où dans ce cas nous aurons une surconsommation de puissance dûe au fait qu'un allumage de trop d'hôtes aura été fait par rapport à la charge demandée par les applications.

La prise en compte du temps de migration sera donc un problème très important, puisqu'en fonction de la variation de la charge, nous pourrons avoir des très bons résultats avec une bonne gestion, ou des résultats contre productifs avec des mauvaises décisions.

3.4 Les différents problèmes

Maintenant que toutes les hypothèses nécessaires ont été posées, il faut s'intéresser à la définition du, ou plutôt dans notre cas, des problèmes. Le problème de l'allocation de services efficace en énergie est un problème, qui nécessite d'effectuer une optimisation multicritères. Il faut en effet bien voir que réduire la consommation d'énergie se fait souvent au détriment de la performance des tâches. De la même manière, augmenter la performance des services résultera dans la majorité des cas en une augmentation de la consommation d'énergie.

Alors comment faire pour réduire la consommation d'énergie, tout en augmentant la performance des tâches ? La solution que nous avons choisi ne réside pas dans la résolution de ce problème précis, mais commence par une transformation de ce problème en trois sous problèmes distincts. La raison principale étant qu'il est difficile de calculer le compromis optimal en un temps raisonnable, dû à la nature complexe du problème, et qu'il est difficile en utilisant des heuristiques de classer les allocations résultantes, c'est-à-dire qu'il est difficile de comparer une perte/gain d'énergie et un gain/perte de performance.

Nous allons donc réduire le problème de l'allocation de tâches efficace en énergie en 3 sous problèmes bien distincts.

3.4.1 Optimiser l'énergie à qualité de service minimale : Bounded-Yield

Le premier sous problème que nous définissons consiste à effectuer une allocation de ressources en fixant une qualité de service, c'est-à-dire fixer une quantité de performance à atteindre, tout en réduisant la consommation d'énergie.

Nous réduisons ainsi le problème à un problème mono objectif, l'objectif étant la réduction de la consommation d'énergie de l'infrastructure, en ajoutant une nouvelle contrainte : tous les services doivent avoir au minimum une certaine qualité de service.

Au final, c'est exactement ce qui pourrait arriver si l'on était dans un cloud possédant des contrats de qualité de service (ou SLA) avec ses clients. Un client se verra garantir 95% du temps ce qu'il avait demandé par exemple, et le fournisseur de services clouds optimisera et gèrera les tâches de manière à réduire ses coûts, notamment en réduisant la consommation de son infrastructure.

Cette nouvelle définition du problème implique, comme mentionné précédemment une réduction du problème multi objectif à un problème mono objectif, réduisant ainsi de façon significative le temps de calcul d'une solution. Par la suite, nous appellerons ce problème BoundedYield.

3.4.2 Optimiser la performance à puissance maximale : BoundedPower

De la même manière que l'on transforme le problème multi objectif en optimisation de l'énergie sous contrainte de performance, on peut facilement imaginer le problème opposé. Celui-ci sera nommé BoundedPower. Il faut par exemple prendre en compte la situation où un propriétaire d'un data center serait confronté à un problème de limitation de l'infrastructure fournissant l'électricité au data center. Il peut arriver une situation où l'infrastructure d'acheminement d'élèctricité jusqu'au data center nécessite d'imposer une limite de consommation d'élèctricité.

On peut aussi facilement imaginer une tarification différente en fonction d'un certain budget de puissance, et ainsi vouloir tenter de rester en deçà de cette limite.

C'est ce qui nous amène à définir le deuxième problème. Ici, il sera question de contraindre l'allocation en terme de puissance consommée par le cluster, tout en optimisant la performance.

Nous aurons donc des situations où par exemple un fournisseur de services ne voudra pas que son infrastructure dépasse les 10 kW de consommation, tout en maximisant la performance des tâches s'exécutant dans son infrastructure. Nous voudrons dans ce cas donner le plus de satisfaction aux tâches qui s'exécutent sur le système, tout en veillant à rester en dessous d'une certaine contrainte de puissance.

Cette modélisation a les mêmes implications que le problème contraint en performance et permet aussi de réduire le problème à une optimisation mono objectif.

3.4.3 Réduire l'énergie et augmenter la performance : MixedObjective

Le dernier type de situation pouvant survenir concerne le cas où un fournisseur de services, bien que n'ayant pas de contraintes spécifiques ni en énergie, ni en performance, veut réduire la consommation d'énergie de son infrastructure, tout en optimisant la performance des tâches exécutées.

Ici, comme les deux problèmes précédents, nous tenterons de réduire en utilisant une valeur de compromis, une métrique, le problème à une résolution mono objectif. Nous appellerons ce problème MixedObjective par la suite.

Cependant, au contraire des deux problèmes précédents, nous garderons ici une part de multi objectif en tentant d'optimiser le compromis. En fait, même s'il est pertinent d'avoir ce genre d'approche, il faudra bien faire attention au genre de compromis que l'on veut faire. Il convient alors de discuter de la valeur d'une métrique par rapport à une autre.

Ainsi, est-il plus intéressant de perdre 2% de performance pour économiser 200W, ou perdre 4% de performance pour économiser 500W ? La réponse dépendra souvent du contexte, dans quel type de système l'on va se placer, avec quel type d'infrastructure. Il sera ainsi difficile de juger de la qualité du compromis atteint.

De plus, le choix de la valeur du compromis sera problématique. En effet, contrairement aux optimisations précédentes, le choix de cette valeur de compromis sera contraignante à plusieurs égards. Qui devra choisir cette valeur ? Est-ce le fournisseur, qui a une connaissance forte de son infrastructure ? Ou est-ce le client, qui lui aura une idée précise de ce qu'il attend, et de ce qu'il sera prêt à sacrifier comme performance ?

Enfin, il faut définir comment fixer cette valeur ainsi que ce qu'elle représentera. On voudra rechercher une valeur représentant quelque chose de concret, comme pouvoir dire "favoriser autant les économies d'énergie que la performance". Dans notre cas, la quantification de l'énergie et celle des performances sera différente, réduisant ainsi l'interprétation de cette valeur de compromis.

Ce problème restera tout de même pertinent dans le sens où aussi bien le client que le fournisseur peuvent vouloir réduire leurs coûts, sacrifiant ainsi un peu de performances pour réduire de l'énergie et économiser de l'argent pour le fournisseur, ainsi que pour le client qui payera moins cher pour des tâches exécutées de manière plus souple.

3.5 Techniques de Résolution

Maintenant que les problèmes que nous allons étudier sont posés, il convient de parler des techniques de résolution de ces problèmes, puisque nous pouvons résoudre les problèmes de plusieurs manières, et les contraintes posées par ces méthodes vont en partie définir leur qualité.

3.5.1 Optimal

La première intuition lors de la résolution d'un problème est celle de tenter d'être capable de calculer la solution optimale de celui-ci. En effet, si cela est possible, nous choisirons toujours de prendre une solution optimale. De plus, s'il est difficile de calculer la solution optimale, et que nous ne pouvons la calculer que dans certains cas, il sera toujours intéressant de l'avoir, notamment pour être capables de comparer les différentes solutions apportées, et ainsi juger de leur efficacité.

Dans notre cas, pour résoudre de façon optimale le problème, nous utiliserons une résolution par programmation linéaire. Cela veut dire que dans un premier temps, nous devrons modéliser le problème comme un ensemble d'équations linéaires représentant des contraintes, puis nous définirons les objectifs en fonction des différents problèmes, nous serons alors capables en utilisant un programme de résolution de calculer la solution optimale, à partir des paramètres d'entrée et sous contraintes.

Il s'avère que la manière dont nous définissons le problème nous amène à avoir un ensemble de contraintes contenant à la fois des entiers et des nombres rationnels. Il est plus complexe de calculer une solution à un problème contenant des entiers, car l'espace de recherche de celui-ci n'est plus continu mais discret.

3.5.2 Complexité du problème

La formulation linéaire contenant des entiers dans notre cas est justifiée par plusieurs variables binaires qui représenteront par exemple l'état allumé ou éteint d'un matériel. Or, un problème d'optimisation mixte entiers et rationnels est connu pour être plus complexe qu'en rationnel.

Plus spécifiquement, le problème de l'allocation de tâches peut être considéré comme un problème de bin packing, c'est-à-dire consistant à placer des volumes (ici les tâches) dans des conteneurs (ici les hôtes). Le problème de bin-packing est un problème d'optimisation NP-complet, ainsi, notre problème d'allocation de ressources efficace en énergie, tel que décrit précédemment,

est une extension du problème de bin packing. Par conséquent, le problème de l'allocation efficace en énergie est aussi NP-complet[47].

Cela veut dire que le calcul des solutions pour ce problème sera en temps exponentiel dans le pire cas. Il faudra donc utiliser, pour le résoudre de façon utilisable, des algorithmes d'approximations heuristiques, permettant de calculer une solution rapidement, avec l'espoir que celle-ci soit proche de l'optimal. Nous pouvons voir avec la figure 3.2, qui représente le temps de calcul moyen de l'optimal pour des instances allant de 4 machines et 8 services à 8 machines et 16 services. Nous pouvons y voir clairement l'augmentation du temps de calcul lié à l'explosion de l'ensemble des combinaisons possibles.

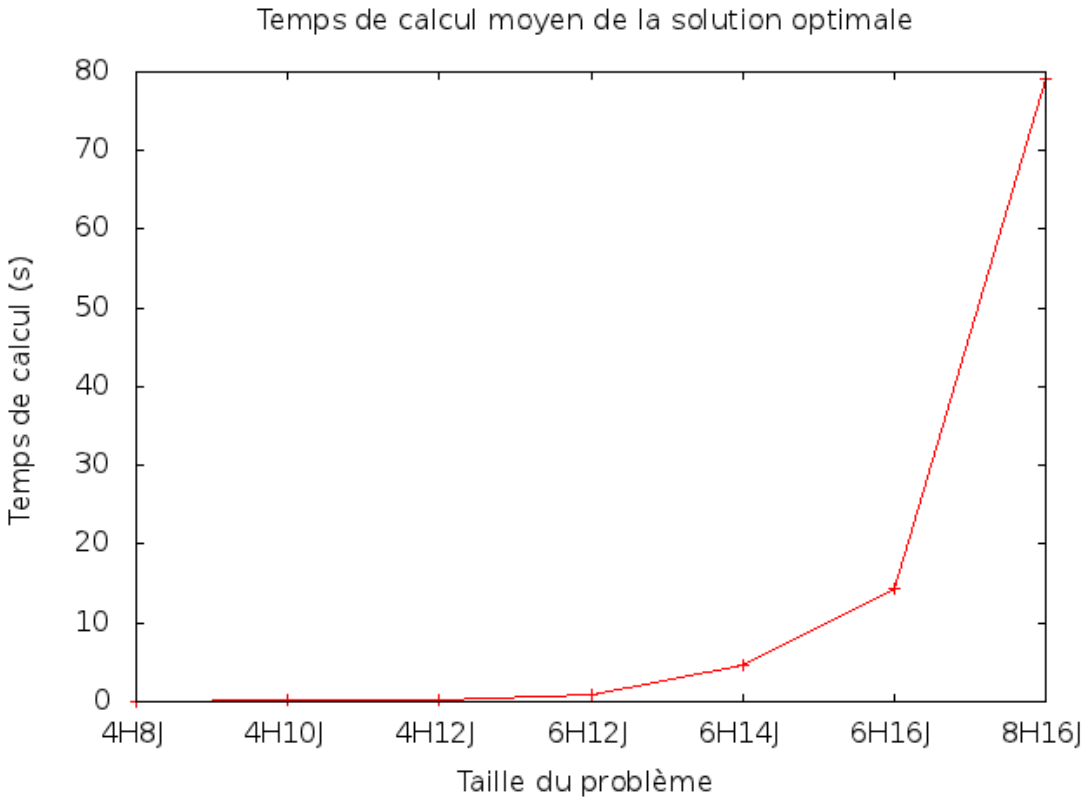


FIGURE 3.2 – Temps de calcul moyen de l'optimal pour une instance en fonction de la taille du problème, la taille étant en nombre d'hôtes(H) et nombre de tâches(J)

3.5.3 Borne rationelle

Lorsque l'on développe des heuristiques et autres algorithmes d'approximation pour résoudre un problème, il faut pouvoir juger de la qualité des résultats fournis par ces approches, et il faut pouvoir les comparer. Pour ce faire, l'approche habituelle est celle consistant à se comparer à la solution optimale. Si l'on peut juger de la distance moyenne d'un algorithme à l'optimal, on peut être capable de répondre aux deux questions : "cet algorithme est-il efficace ?" et "cet algorithme est-il plus efficace qu'un autre ?".

Dans notre cas, il est possible au vu des techniques de résolution existantes de trouver la solution optimale en un temps raisonnable, mais seulement pour des petites instances. Comme mentionné précédemment, il est possible pour des instances petites et de taille moyenne de calculer la solution optimale et ainsi s'y comparer. On peut alors se demander si comparer des algorithmes sur la base unique des petits problèmes ne serait pas suffisant ?

Il est en effet intéressant de comparer les résultats des algorithmes aux valeurs optimales, mais cela ne suffit pas. Il faudrait présupposer qu'il n'y a aucun effet induit par le passage à l'échelle. C'est-à-dire que lorsque l'on augmente la taille des instances, les effets et les différents cas sont strictements les mêmes. Seulement cela n'apparait pas être le cas. Par exemple les petites instances de notre problème ont un espace de combinaison possible restreint, ce qui restreint l'action possible par les algorithmes, alors qu'une instance large possèdera un large espace de combinaison possible, ce qui induit une plus grande facilité à s'approcher des bonnes solutions, puisqu'il y a plus de choix possibles. Ainsi, comme nous le verrons par la suite dans le chapitre 4, on pourra facilement avoir un algorithme qui fonctionnera bien mieux à large échelle, dans la mesure où plus de choix lui est proposé, et qu'il y a plus de possibilités que lorsque l'instance est petite, et ainsi plus de capacité d'avoir des bons résultats.

Pour palier à ce problème, l'expression linéaire des contraintes de notre problème nous permet de calculer la solution rationnelle optimale, qui correspond au passage des variable binaires ou entières en variables rationnelles. Cette borne rationnelle est une borne supérieure à la valeur optimale, puisque des variables rationnelles sont plus permissives et autorisent l'allocation de fractions de services, ou l'allumage de portions de serveurs. Cette borne rationnelle est intéressante à calculer justement parce qu'elle est une borne supérieure sur la solution optimale, et qu'elle se calcule en temps polynomial.

Cependant, la solution fournie par la borne rationnelle n'est pas une solution valide. En passant de variables entières à des variables rationnelles, nous nous retrouverons avec une solution comprenant des impossibilité réelles. Par exemple, si nous avons une variable entière qui définit l'état allumé ou éteint d'un hôte, celle-ci passera rationnelle, et ainsi nous pourrons très bien avoir une solution comprenant des hôtes à moitié allumés, ce qui n'est pas cohérent avec la réalité.

Pourtant, pour les grandes instances, nous pourrons, même s'il n'est pas vraiment possible de juger de la distance entre la borne rationnelle et la solution optimale, comparer les algorithmes entre eux en comparant leur distances respectives ainsi qu'à la borne rationnelle.

3.5.4 Algorithmes d'approximation

Pour terminer, nous devons définir les heuristiques et comment nous allons implémenter des algorithmes d'approximation afin d'étudier les effets des différents types d'allocation que nous pouvons avoir. Commençons par définir les différents types d'algorithmes que nous serons amenés à utiliser. Les différents algorithmes que nous serons amenés à manipuler sont des algorithmes dits de *bin-packing*, qui consistent à placer des volumes dans des conteneurs, afin de minimiser le nombre de conteneurs utilisés. Nous utiliserons ce type d'algorithmes car le problème d'allocation de services sur des serveurs se réduit directement à un problème de bin-packing. Les serveurs étant les conteneurs, et les services étant des volumes pour lesquels chaque ressource correspond à une dimension. Le but étant d'avoir une allocation efficace, autrement dit d'avoir tous les services alloués sur un nombre réduit de serveurs. La différence étant que l'on peut compresser les valeurs en diminuant la qualité de service allouée aux services.

First Fit

L'algorithme de type First Fit est un algorithme glouton très connu et très simple, qui consiste tout simplement à allouer chaque volume dans le premier conteneur sur lequel le volume va

rentrer. L'algorithme 1 présente en pseudo code l'algorithme First Fit. On notera ici qu'il est possible que l'algorithme First Fit, de type glouton, puisse échouer.

Si l'on regarde la manière dont est alloué chaque volume, on peut voir qu'il arrive des cas où l'on ne trouvera pas de conteneur pouvant contenir le ou les volumes restant à allouer, amenant ainsi l'algorithme à échouer. Cela ne voudra pourtant pas dire qu'il n'est pas possible de trouver une allocation permettant à tous les volumes d'être alloués. L'algorithme dit de First Fit fait

Algorithm 1 L'algorithme First Fit

```
Require: Liste de conteneurs triée
  for all v ∈ volumes do
    for all c ∈ conteneurs do
      if c peut contenir v then
        Allouer v sur c
      else if Tous les conteneurs n'ont pas été examinés then
        Passer au conteneur suivant
      else
        ECHEC
      end if
    end for
  end for
```

partie de la grande famille des heuristiques gloutonnes que nous appellerons "*-Fit". Celle-ci fonctionne de la même manière que le First Fit, et va tenter d'allouer chaque volume l'un après l'autre, de manière à ce que ceux-ci soient conformes à la règle selon laquelle ils doivent avoir étés alloués. Par exemple, nous avons pour l'algorithme First Fit la règle disant que chaque volume doit être alloué sur le premier conteneur où il rentre. Pour l'algorithme de type Best Fit par contre, nous avons chaque volume devant être alloué sur le conteneur dont l'utilisation sera la plus grande après allocation, ou autrement dit, là où l'espace pouvant contenir le volume est le plus petit.

La famille des "*-Fit" contient aussi des algorithmes tels que Worst Fit, Better Fit, ainsi que tout autre algorithme fonctionnant de la même manière.

De ce fait, nous avons la capacité de connaitre les tendances et le fonctionnement de l'algorithme First Fit par exemple. L'algorithme est suffisamment connu pour pouvoir avoir connaissance du temps qu'il prend, mais aussi des tendances d'allocation de tels algorithmes. Il faudra bien comprendre qu'un algorithme naif tel que First Fit ne va pas avoir de bonnes performances, du simple fait qu'il sera très dépendant de l'ordre dans lequel on affectera les volumes. En fait, il sera aussi dépendant de l'ordre dans lequel on va choisir les conteneurs, tout simplement car il ne prendra pas en compte ni l'utilisation courante des conteneurs, ni la taille des volumes, car il essaiera de trouver une allocation où chaque volume va sur le premier conteneur trouvé.

Ainsi, afin d'évaluer le résultat d'une allocation First Fit, il faudra comprendre que l'effet de consolidation apporté par un algorithme First Fit sera présent, mais que celui-ci sera dans la majorité des cas relativement loin de la consolidation optimale.

Nous pourrons donc comparer d'autres algorithmes à ceux de type *-Fit, tout en gardant à l'esprit la manière, simple, dont ils fonctionnent.

Pour prendre en compte la consommation de puissance dans les différents algorithmes de type *-Fit, afin de les rendre un peu plus efficaces énergétiquement parlant, il faudra agir sur deux points non pris en compte par l'algorithme *-Fit de base. Premièrement comme mentionné précédemment, il faudra prendre en compte l'ordre des volumes.

Comme des études précédentes l'ont montrés (par exemple dans [100]), l'ordre d'allocation des

volumes joue un rôle crucial dans le succès de l'allocation d'une part (allouer les volumes les plus gros en premier résulte généralement en une augmentation du taux de succès d'un algorithme), mais aussi dans le taux de consolidation d'autre part (c'est-à-dire à quel point les volumes sont regroupés sur un petit sous-ensemble de conteneurs). De fait, si nous avons des algorithmes qui ont plus de succès, et qui allouent sur un nombre plus petits d'hôtes, nous aurons comme effet collatéral une consommation de puissance plus faible.

Deuxièmement, il faudra prendre en compte l'ordre d'allocation des hôtes pour diminuer l'énergie. Cela étant, même si nous ne nous plaçons pas dans un contexte d'hôtes homogènes (autrement dit de conteneurs de taille égale), puisque nous sommes sur un même cluster ici, les machines sont considérées toujours comme hétérogènes en énergie. Ainsi, il devient important d'allouer les tâches sur les hôtes les plus efficaces énergétiquement.

De ces différentes manières, nous pourrons améliorer de façon simple les algorithmes gloutons de type *-Fit, pour les faire devenir plus pertinents vis-à-vis de la consommation d'énergie, tout en gardant leurs fonctionnements primaires, et en étant ainsi capables de garder une comparaison éclairée (cf. section 4.3).

Round Robin

L'algorithme de type Round Robin, est un autre algorithme au fonctionnement connu. Il consiste tout simplement à tenter d'allouer chaque volume dans un conteneur différent.

L'algorithme Round Robin peut échouer s'il ne trouve plus la place d'allouer les volumes restants. Son fonctionnement est simple, comme nous pouvons le voir en 2

Algorithm 2 L'algorithme Round Robin

```
c_depart = 1
for all v ∈ volumes do
    for all c ∈ conteneurs en commençant par c_dEpart do
        if c peut contenir v then
            Allouer v sur c
            c_depart = c + 1
        else
            Passer au conteneur suivant
        end if
        if c = c_depart then
            ECHEC
        end if
    end for
end for
```

L'algorithme Round Robin est un algorithme réputé pour être équitable. Effectivement, dans la mesure où celui-ci tente d'allouer chaque volume à un conteneur différent, nous sommes en mesure de garantir une équité dans l'attribution des volumes.

De fait, on peut donc se poser la question de savoir si ce genre d'algorithme est efficace énergétiquement parlant. La réponse sera bien évidemment que, tel quel, les algorithmes de type Round Robin ne sont que peu efficaces énergétiquement parlant. On pense bien qu'allouer une tâche à chaque hôte n'est pas le meilleur moyen de réduire le nombre d'hôtes utilisés, et par conséquent la puissance consommée. Par contre, nous aurons souvent une qualité de service excellente, puisque si l'algorithme réussit, nous aurons chaque tâche allouée avec au moins ce qu'elle avait requis, et de plus, une interférence inter tâche minimisée, puisque chaque tâche sera

sur son hôte avec un nombre minimum d'autres tâches.

Il apparaitra donc que dès que nous aurons plus de tâches que d'hôtes, nous aurons tous les hôtes allumés, consommant ainsi le plus d'énergie. Ainsi, nous pourrons tout de même imaginer des moyens afin de réduire l'énergie du système. Par exemple en agissant sur le nombre (et le choix) des hôtes proposés à l'algorithme, nous pourrons injecter des concepts d'économies d'énergie dans un algorithme qui ne l'est pas, le rendant ainsi un peu plus efficace.

Nous pourrons ainsi avoir un algorithme basé sur le round robin qui aura pour but d'allouer les tâches aux hôtes en tentant de choisir un nombre minimum d'hôtes, permettant ainsi une bonne efficacité énergétique (cf. section 4.3).

Vector Packing

Les algorithmes de type vector packing sont une extension des algorithmes dits de "bin packing" à plusieurs dimensions. Si le bin packing est à une dimension, il correspond au fait d'affecter des segments à l'intérieur d'un conteneur (un autre segment plus grand). De la même manière, le bin packing à 2 dimensions tentera d'allouer des surfaces dans d'autres surfaces. Pour 3 dimensions nous parlerons donc de volumes. En généralisant, le vector packing désignera les algorithmes permettant d'affecter des vecteurs dans des conteneurs. Dans notre cas si nous prenons deux dimensions, le CPU et la RAM, nous aurons à allouer des tâches représentées par des vecteurs (cpu, mem) dans des hôtes représentés par des vecteurs (capacité_CPU, capacité_MEM).

On peut donc se demander pourquoi nous ferons la distinction entre les algorithmes gloutons de type first fit et les algorithmes de vector packing. La raison première se situe dans le choix des tâches à allouer. Souvent, on appellera un algorithme de vector packing un algorithme possédant une heuristique de choix à la fois des hôtes, mais aussi des tâches. Comme nous nous placerons dans un contexte multi dimensionnel, nous aurons des choix à faire à la fois dans les tâches, (c'est-à-dire quelle tâche choisir à quel moment de l'allocation, par exemple prendre une tâche à forte contrainte mémoire ou non), dans les hôtes, et pourquoi pas dans les différents clusters.

Dans nos travaux, nous nous intéresserons à un algorithme de vector packing en particulier décrit dans [73], basé sur une allocation qui permet à chaque tâche de contrebalancer le déséquilibre en consommation de ressource à un instant donné de l'exécution de l'algorithme, qui a prouvé donner des bons résultats autant dans le taux de succès que dans l'allocation résultante. Nous l'avons adapté à l'économie de manière simple dans un premier temps, en lui faisant choisir les hôtes les plus efficaces en premier.

Le principe est décrit en pseudo algorithme en 3. Celui-ci correspond à celui utilisé dans l'algorithme EAResAlloc

La figure 3.3 illustre les différents fonctionnements des différents algorithmes d'approximation. Nous voyons que l'algorithme FirstFit alloue les tâches dans l'ordre, l'algorithme RoundRobin alloue une tâche sur chaque hôte, et l'algorithme VectorPacking alloue en contrebalançant le déséquilibre de ressource à chaque instant de l'allocation.

L'importance des tris

On remarquera que chaque algorithme peut de base être amélioré en incorporant un tri, que ce soit au niveau des tâches ou des hôtes, afin d'avoir des meilleures performances. Ces tris ont beaucoup d'importance, car ils vont faire la différence entre un algorithme FirstFit naïf et un algorithme un peu plus "intelligent". Nous aurons par exemple le tri des tâches en prenant en premier ceux qui ont les plus grosses demandes en ressources, ce qui aura pour effet d'améliorer à la fois le taux de succès de l'algorithme, c'est-à-dire qu'il va moins souvent échouer, mais aussi de résulter en de meilleures allocations. Les tris permettant de commencer par les grosses demandes

Algorithm 3 L'algorithme Vector Packing

```
Trier les volumes selon une relation d'ordre prédéfinie
Séparer les volumes en 2 listes :
Liste 1 = volumes ayant une dimension d1 plus importante que la dimension d2
Liste 2 = volumes ayant une dimension d2 plus importante que la dimension d1
Trier les conteneurs
for all c ∈ conteneurs do
    while Un volume peut être alloué sur c do
        Choisir la liste permettant de contrebalancer le déséquilibre d'utilisation de ressource courant du conteneur
        Choisir le premier volume dans cette liste qui rentre sur c
    end while
    if Il reste des volumes à allouer then
        if conteneur_suivant existe then
            h ← conteneut_suivant
        else
            ECHEC
        end if
    else
        SUCCES
    end if
end for
```

Tri	Minimum Yield
Rigide en premier	0.828970
Fluide en premier	0.821400

TABLE 3.1 – Différence de résultat pour l'algorithme First Fit selon différents tris de tâches

en ressources auront le plus souvent de meilleurs résultats que ceux commençant par les plus petites demandes [100].

Le tableau 3.1 montre un exemple de l'impact que peuvent avoir les différents tris de tâches sur certains algorithmes.

De la même manière, nous pouvons agir sur les tris des hôtes. Ici, nous parlerons plutôt de capacité (en terme de taille, et ainsi de qualité de services), ou d'efficacité (par exemple efficacité énergétique). Si l'on prend le cas de l'efficacité énergétique, qui va nous intéresser dans le cadre d'un mono cluster homogène en capacité, mais hétérogène en puissance, nous pourrons voir comme dans le tableau 3.2 l'importance que peut revêtir le choix d'hôtes efficaces en premier. Par conséquent, le tri des hôtes sera une étape simple, mais efficace dans le chemin vers les économies d'énergie. Il faudra que pour chaque algorithme ces différents tris soient présents, mais aussi pris en compte dans les résultats.

Tri	Puissance consommée (Watts)
Petit C^{max} en premier	2980.47
Grand C^{max} en premier	3608.25

TABLE 3.2 – Importance du tri des hôtes sur la consommation d'énergie

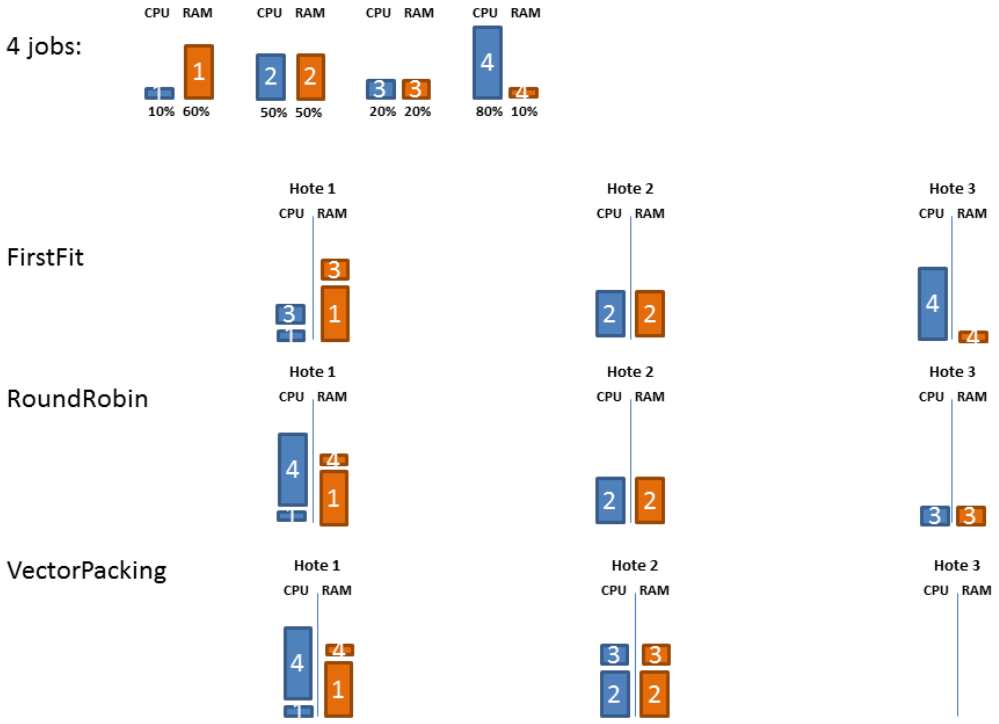


FIGURE 3.3 – Exemple d'allocation des différents algorithmes d'approximation

Chapitre 4

Allocation statique

Le problème de l'allocation de tâches étant complexe, nous avons choisi de le séparer en deux problèmes distincts, en prenant pour commencer l'allocation statique, où nous allouons les tâches avant leur exécution, et l'allocation dynamique, qui décrira ce qu'il se passe après, pendant l'exécution des tâches.

4.1 Description

4.1.1 Contexte

Comme mentionné précédemment, la gestion de tâches nous permet de gérer à plusieurs niveaux les tâches s'exécutant sur notre infrastructure, mais aussi de se placer dans un contexte temporel ou non. Lorsque l'on va alors parler de l'allocation statique des tâches, nous nous plaçons en amont de l'exécution des tâches. Cela peut correspondre au moment où l'on démarre les serveurs, par exemple après une maintenance des serveurs : avant le début de l'exécution des tâches, on peut planifier, c'est-à-dire décider où et comment nous allons exécuter les tâches.

Par conséquent, cela présupposera une connaissance suffisante des tâches à exécuter afin de pouvoir prendre des décisions cohérentes. Par exemple, nous pourrons définir les besoins d'une tâche en prenant en compte l'exécution des jours précédents.

De plus, il faudra comme mentionné précédemment que les tâches soient suffisamment stables pour que l'allocation, qui n'est faite qu'une fois pour l'exécution, soit efficace.

Ainsi, réduire le problème à l'allocation statique nécessite une définition du problème des tâches en tant que besoins prédéfinis avant l'exécution de la tâche.

Nous pouvons prendre l'allocation statique comme une pseudo allocation dynamique, c'est-à-dire que nous pouvons faire une allocation de toutes les tâches depuis un système vide et de manière périodique. Pour cela, il faudra supposer que la période de réallocation soit suffisamment grande pour pouvoir négliger les coûts de migration. De la même manière ici, nous pourrions avoir des migrations de tâches, mais celles-ci seraient considérées comme instantanées, dans la mesure où la durée de migration est grandement inférieure à la durée totale des tâches.

Ainsi nous aurons un système où notre but sera de faire une allocation efficace des tâches, dont les besoins seront bien définis, des tâches sur les machines, tout en étant en accord avec les contraintes fixées. Nous relacherons ensuite certaines de ces contraintes dans les chapitres suivants.

4.1.2 Hypothèses

Ayant restreint le problème à une allocation statique, nous pouvons nous poser la question de la validité des hypothèses posées précédemment dans le chapitre 3. En effet, certaines hypothèses tiennent exactement dans la manière dont nous les avons posées, mais d'autres doivent être modifiées quelque peu, voire supprimées en fonction. Il faudra aussi bien penser que l'allocation statique nécessite que les hypothèses et les données des différents problèmes soient valides pour la durée de l'allocation statique.

C'est le cas par exemple de l'hypothèse de la migration, qui ici est présumée instantanée. Nous ignorerons ainsi les effets de la migration, comme le temps qu'elle met, ainsi que son coût, car l'allocation statique présuppose une durée suffisamment longue d'exécution pour nous permettre d'ignorer les effets négatifs de cette hypothèse.

L'hypothèse concernant le type de tâches que nous prenons en compte doit être relachée pour devenir des tâches de type service uniquement, ou des tâches de calculs suffisamment longues mais finies qui peuvent être allouées de manière statique de façon cohérente, c'est-à-dire suffisamment longues pour permettre les autre hypothèses, mais aussi suffisamment stables pour que l'allocation soit valide tout au long de l'exécution. Cela est aussi en relation avec l'hypothèse présumant que les tâches sont suffisamment constantes pour que l'allocation soit valide pendant la durée de l'exécution. Ainsi, cette hypothèse tient ici, dans la mesure où une allocation statique nécessite des tâches constantes.

Nous présumerons aussi que l'allumage et l'extinction des hôtes est possible, mais sans effets négatifs. De la même manière que pour la migration, nous supposerons que les hôtes inutilisés seront éteints et les hôtes utilisés seront allumés, mais que les effets négatifs liés à l'allumage/extinction des hôtes, tels que le temps d'allumage, sont ignorés, du fait de la différence d'échelle de temps.

De la même manière que pour l'allumage et l'extinction des hôtes, nous présumerons que l'allumage et l'extinction des infrastructures de refroidissement se fait une fois que l'allocation est faite, et qu'une fois éteint, nous n'aurons pas à les rallumer avant la prochaine allocation.

Enfin, nous supposerons que nous avons à l'intérieur d'un cluster des hôtes homogènes en puissance de calcul, mais hétérogènes en consommation d'électricité.

4.1.3 Leviers utilisables

Regardons maintenant les leviers que nous pourrons ou devrons utiliser lors de l'allocation statique.

Le levier de la virtualisation est bien sûr un choix évident, dans la mesure où nos besoins de tâches et leur allocation seront exprimés en pourcentage de l'hôte sur lequel elles sont allouées. Cette fonctionalité est rendue possible par la virtualisation, et bien que nous ne prendrons pas en compte la migration pour l'allocation statique, nous utiliserons le levier de la virtualisation pour effectuer notre allocation.

Nous n'utiliserons par contre pas la technique de DVFS, car celle-ci comme mentionné précédemment possède des inconvénients. De plus, si l'on met le DVFS en relation avec l'hypothèse d'hétérogénéité des hôtes, nous pourrions définir les différents états de DVFS comme des hôtes exclusifs. Cependant, bien que la gestion du DVFS présente des avantages, ne pas l'utiliser ici est surtout dû au fait que nous n'aurons pas pour l'allocation statique de gestionnaire de tâches, qui permettrait de définir à la volée la fréquence à laquelle doivent s'exécuter les tâches. Enfin, nous avons aussi l'échelle différente de temps entre les tâches qui sont suffisamment longues pour pouvoir ne pas prendre en compte les temps d'allumage et d'extinction des hôtes, et la micro gestion nécessaire pour faire un DVFS efficace, faisant que nous ne prendrons pas en compte ce levier, pour éviter les effets indésirables entre l'allocation et la gestion des fréquences.

Enfin, l'allumage et l'extinction des machines est au coeur de l'efficacité des algorithmes. Une grosse partie des économies d'énergie que nous allons faire sera due au fait que nous pourrons éteindre les hôtes inutilisés, ce qui implique que ce levier sera donc important à prendre en compte.

4.2 Programme Linéaire

Afin de bien étudier le système, et de pouvoir trouver l'allocation optimale, nous devons tout d'abord modéliser le problème de manière formelle. Pour cela, nous allons utiliser la programmation linéaire.

La programmation linéaire est un outil utilisé dans l'optimisation qui permet de décrire un problème comme un ensemble de contraintes linéaires. Elle permet d'utiliser des méthodes de résolutions comme par exemple la méthode du simplex, afin de calculer la solution optimale au problème décrit. Puisque les méthodes d'optimisation ont une complexité équivalente, nous avons choisi d'utiliser celle-ci qui présente l'avantage de n'utiliser qu'un ensemble de contraintes linéaires.

Le formalisme de la programmation linéaire a l'avantage de décrire un problème de manière simple à l'aide de contraintes linéaires.

L'ensemble de contrainte linéaire peut contenir à la fois des variables entières, et des variables rationnelles. Les variables entières augmentant la difficulté de trouver la solution optimale.

Dans cette section, nous allons décrire le programme linéaire décrivant notre problème. C'est-à-dire décrire l'ensemble des contraintes linéaires nous permettant de décrire et borner notre problème. Cet ensemble de contraintes sera ensuite utilisé notamment comme entrée d'un solveur linéaire pour calculer les solutions optimales et rationnelles. La solution optimale prendra bien plus de temps à calculer mais sera fidèle au modèle, alors que la solution rationnelle sera une solution approchée qui se calculera en temps polynomial.

Dans notre cas, nous aurons un problème mixte entier et rationnel, car pour décrire l'état allumé ou éteint d'un hôte, nous ne pourrons pas utiliser une variable rationnelle (une machine ne peut être à moitié allumée).

4.2.1 Ensemble de contraintes

Afin de décrire les contraintes permettant de traduire les hypothèses en un programme linéaire, commençons par définir les variables et notations, qui seront récapitulées dans le tableau 4.1. Rappelons ici que nous nous plaçons dans le contexte d'un cluster de calcul et que l'on néglige les infrastructure connexes comme par exemple les infrastructures réseau.

Un cloud est composé de K clusters, chaque cluster comportant H_k hôtes. Nous prendrons en compte pour les hôtes deux types de ressources. La première sera fluide, c'est-à-dire que nous pourrons la modifier sans nuire à l'intégrité du programme (comme le CPU ou la bande passante dans certains cas), et la seconde sera rigide, ce qui veut dire que nous ne pourrons pas allouer un montant inférieur à ce que la tâche requiert (comme la RAM, l'espace disque). Ici, bien que nous puissions prendre en compte de multiples ressources, nous ne prendrons, pour simplifier qu'une ressource de chaque type, le CPU (exprimé en Millions d'Instructions Par Secondes ou MIPS) comme ressource fluide et la RAM (exprimée en Mégaoctets ou Gigaoctets) comme ressource rigide. Des travaux existants ont déjà effectué l'extension à plusieurs ressources de chaque type [100].

Le but de l'allocation sera de placer les J tâches sur les H hôtes, sous les contraintes que nous allons définir.

Nous définissons donc les variables suivantes :

$$\forall j, h \quad e_{jh} \in \{0, 1\} \tag{4.1}$$
$$\forall h \quad p_h \in \{0, 1\} \tag{4.2}$$
$$\forall h, k \quad b_{hk} \in \{0, 1\} \tag{4.3}$$
$$\forall j, h \quad \alpha_{jh} \in \mathbb{Q} \tag{4.4}$$
$$\forall j, k \quad r_{jk} \in \mathbb{Q} \tag{4.5}$$
$$\forall j, k \quad m_{jk} \in \mathbb{Q} \tag{4.6}$$

La tâche est représentée par j, l'hôte est représenté par h et le cluster est représenté par k.

e_{jh} est une variable binaire qui permet de décrire le mapping de la tâche j sur l'hôte h. Si $e_{jh} = 1$ la tâche j est allouée à l'hôte h.

p_h est aussi une variable binaire qui permet de décrire l'état de l'hôte h. Si $p_h = 1$ l'hôte h est allumé, si $p_h = 0$, l'hôte h est éteint.

b_{hk} est une variable binaire permettant de décrire l'appartenance d'un hôte h à un cluster k. Cela veut dire que si $b_{hk} = 1$ l'hôte h appartient au cluster k. Cette variable ne sera pas une variable du programme linéaire, mais une valeur d'entrée. En effet, nous saurons à l'avance quel hôte appartient à quel cluster, et cela ne fera pas partie des décisions prises par le programme de décision.

Les variables rationnelles r_{jk} et m_{jk} représentent le montant en pourcentage de ressource fluide (respectivement rigide) que la tâche j requiert sur le cluster k. Ici, nous aurons donc r_{jk} représentant le pourcentage de CPU que la tâche j demande sur le cluster k, et de la même manière m_{jk} le pourcentage mémoire requis par la tâche j sur les machines du cluster k. Il convient ici de rappeler que nous sommes contraints par l'hypothèse d'homogénéité intra cluster, ce qui veut dire qu'une tâche aura le même pourcentage sur toutes les machines d'un même cluster, mais que cette valeur sera différente entre les différents clusters du cloud.

Enfin, nous avons la variable α_{jh} représentant le montant de ressource fluide effectivement alloué à la tâche j sur l'hôte h. Notons ici que nous n'aurons pas de variable similaire pour la mémoire, celle-ci étant considérée rigide, et par conséquent ne pouvant pas différer de m_{jk}.

$$\forall j \quad \sum_{h=1}^{H} e_{jh} = 1 \tag{4.7}$$

La première contrainte que nous allons prendre en compte est représentée par l'équation 4.7. Celle-ci concerne la matrice e de l'exécution des tâches sur leurs hôtes. Ici, quelque soit la tâche j, la somme des valeurs du vecteur e_j doit être égale à 1. Ce qui veut dire que quelque soit la tâche, celle-ci doit être allouée à un seul et unique hôte, et donc qu'une tâche ne peut pas être allouée à plusieurs hôtes en même temps.

$$\forall h \quad \sum_{k=1}^{K} b_{hk} = 1 \tag{4.8}$$

L'équation 4.8 donne une contrainte sur l'appartenance d'un hôte à un cluster. Un hôte ne peut appartenir qu'à un seul et unique cluster, ce qui implique que nos clusters sont des ensembles d'hôtes disjoints. Comme mentionné précedemment, la variable b_{hk} est une entrée du système et n'influe pas sur la décision. Par conséquent, cette contrainte servira juste à garder la consistance du système.

Variable	Paramètre d'entrée	Type	Description
J	oui	Entier	Nombre de tâches
H	oui	Entier	Nombre d'hôtes total
H_k	oui	Entier	Nombre d'hôtes du cluster k
K	oui	Entier	Nombre de clusters
L	oui	Entier	Nombre d'équipements de refroidissement
C_h^{min}	oui	Entier	Consommation en watts de l'hôte h lorsqu'il est inactif mais allumé
C_h^{max}	oui	Entier	Consommation en watts de l'hôte h lorsqu'il est chargé à 100%
e_{jh}	non	Binaire	La tâche j est sur l'hôte h
p_h	non	Binaire	L'hôte h est allumé (=1) ou éteint (=0)
b_{hk}	oui	Binaire	L'hôte h appartient au cluster k
ce_{kl}	non	Binaire	L'équipement de refroidissement l du cluster k est allumé
c_{hl}	oui	Binaire	l'hôte h est refroidi par l'équipement de refroidissement l
α_{jh}	non	Rationnel	Fraction de CPU alloué à j sur l'hôte h
r_{jk}	oui	Rationnel	Fraction de CPU demandé par la tâche j sur le cluster k
m_{jk}	oui	Rationnel	Fraction de RAM requis par la tâche j sur le cluster k

TABLE 4.1 – Résumé des notations

$$\forall j, h \quad \alpha_{jh} \leq \sum_{k=1}^{K} (r_{jk} \times b_{hk}) \tag{4.9}$$

La contrainte 4.9 définit la limite supérieure de ce que l'on pourra allouer aux tâches j sur les différents hôtes des clusters. Une tâche j ne pourra pas se voir allouer sur un hôte h du cluster k plus que le pourcentage de ressource fluide qu'elle avait demandé sur les hôtes de ce cluster.

$$\forall j, h \quad 0 \leq \alpha_{jh} \leq e_{jh} \tag{4.10}$$

De la même manière la contrainte 4.10 pose aussi une borne supérieure sur l'allocation maximale des tâches sur les hôtes, à savoir que celles-ci ne peuvent dépasser 100% de l'hôte. De plus, cette équation contraint aussi de manière à ce qu'une tâche j ne puisse consommer des ressources que sur l'hôte sur lequel elle est allouée. Si la tâche j est allouée sur l'hôte h, alors $e_{jh} = 1$, et ainsi nous avons $\alpha_{jh} \leq 1$, sinon nous avons $\alpha_{jh} \leq 0$, soit 0.

$$\forall h \quad \sum_{j=1}^{J} \alpha_{jh} \leq p_h \tag{4.11}$$

Prenons ensuite la contrainte 4.11. Cette contrainte sert à dire qu'une tâche ne peut consommer des ressources seulement sur un hôte allumé. Cela veut aussi dire que l'on ne pourra évidemment pas avoir de tâches consommant des ressources sur un hôte éteint.

$$\forall h \quad p_h \leq \sum_{j=1}^{J} e_{jh} \tag{4.12}$$

La suite logique est donc la contrainte 4.12, qui définit qu'un hôte allumé implique qu'au moins une tâche s'exécute dessus. Avec de telles contraintes, il est intéressant de noter que l'allumage et l'extinction des hôtes est considéré comme instantané.

$$\forall h, k \quad \sum_{j=1}^{J} (e_{jh} \times b_{hk} \times m_{jk}) \leq p_h \tag{4.13}$$

Enfin, la contrainte 4.13 définit de la même manière que la contrainte 4.9, la consommation de ressource rigide (ici la RAM) de la tâche j sur l'hôte h du cluster k. Plus précisément, cette équation définit que si une tâche s'exécute sur un hôte, celui-ci est allumé et la tâche consomme le pourcentage de mémoire qu'elle avait requis sur cet hôte.

Toutes ces contraintes linéaires posent la base du modèle pour l'allocation de tâches sur un ensemble de clusters homogènes. Ici, nous avons étendu les contraintes définies dans [100], pour y intégrer l'hétérogénéité inter cluster, ainsi que la notion même de clusters. Il nous faut maintenant intégrer à ce modèle les infrastructures de refroidissement.

Pour ce faire, il convient tout d'abord de rappeler le modèle de refroidissement que nous avons choisi et sa consommation. Dans les clouds, et plus généralement dans le monde des infrastructure de calculs type cluster, les configurations spatiales des machines, ainsi que les machines elles mêmes, sont diverses et variées. Si souvent les conditions environnementales sont différentes, que ce soit pour la place ou pour la circulation de l'air, chaque cluster répond à des contraintes et besoins précis. Ainsi, il est difficile de définir un modèle cohérent et suffisamment générique pour satisfaire les besoins d'un grand nombre de clusters. Nous aurons par exemple des clusters refroidis à l'air à l'aide de ventilation, alors que d'autres le seront à l'aide de l'eau. Nous aurons des clusters avec des flux d'air global bien précis, alors que d'autres seront définis en partie indépendants.

Pourtant, des approximations peuvent être faites, afin d'unifier certaines de ces différentes. Commençons par prendre le modèle énergétique du refroidissement. Celui-ci est dicté par la formule de dissipation de la chaleur, qui n'est pas linéaire comme on peut voir dans [27]. Il faudra donc dans notre cas faire la première approximation qui sera de définir un modèle linéaire de consommation de l'infrastructure de refroidissement.

Ensuite, il convient de parler de la manière dont est refroidi un cluster. Par exemple, est-ce que celui-ci possède un refroidisseur général, ou chaque module possède son propre système de refroidissement ? Dans notre cas nous définirons les clusters comme des ensemble de machines organisées par armoires. Chaque armoire de machines se verra attribuer une infrastructure de refroidissement qui servira à refroidir uniquement ces machines. Il faut préciser que dans une infrastructure réelle, nous aurons des interactions entre les différents systèmes de refroidissement, et que l'air froid rejeté ou non par ceux-ci aura un impact sur la température des autres armoires. Cependant, cette interaction est minime dans beaucoup de clusters. De plus, nous voyons une popularisation des systèmes de type modulaires, dans lesquels chaque module est un container indépendant, possédant à la fois ses machines, son alimentation et son refroidissement. Un parallèle étant à faire avec les clusters ayant un unique module de climatisation, qui va servir à refroidir l'air, et de multiples modules d'acheminement, qui serviront à acheminer l'air froid vers les machines, et/ou à en retirer l'air chaud. Dans ces cas là, le module de conditionnement sera considéré comme un surcoût inévitable sur lequel nous ne pourrons pas agir, alors que les module d'acheminement donneront la possibilité d'être éteints.

Ainsi donc, nous pourrons représenter notre infrastructure comme un ensemble de clusters modulaires. Chaque refroidissement sera défini en fonction du fonctionnement ou non d'au moins une de ses machines.

Au niveau du programme linéaire, nous avons chaque infrastructure de refroidissement (qui

sera dans certains cas uniquement l'infrastructure d'acheminement du froid), qui peut être éteinte ou allumée en fonction de la charge des machines. De plus, nous supposerons que chaque équipement de refroidissement est identique à l'intérieur du cluster, mais diffère entre les clusters.

$$\forall k, l \qquad ce_{kl} \in \{0, 1\} \tag{4.14}$$
$$\forall h, l \qquad c_{hl} \in \{0, 1\} \tag{4.15}$$

Afin de compléter l'ensemble de contraintes, nous définissons des nouvelles notations pour l'infrastructure de refroidissement. Nous aurons $l \in L$, l représentant un équipement de refroidissement, et L le nombre total de ces équipements. Nous définissons ainsi les variables binaires ce_{kl} et c_{hl} (contraintes 4.14 et 4.15). La variable ce_{kl} permet de définir l'état de l'équipement de refroidissement l sur le cluster k, c'est-à-dire $ce_{kl} = 0$ si celui-ci est éteint, $ce_{kl} = 1$ si il est allumé. La variable c_{hl}, qui est quant à elle un paramètre de l'instance, permet de définir que l'hôte h est refroidi par l'équipement de refroidissement l.

$$\forall k, l \quad \sum_{h=1}^{H} (p_h \times c_{hl} \times b_{hk}) \geq ce_{kl} \tag{4.16}$$

La contrainte 4.16 permet de décrire le fait que si aucun hôte refroidi par l'équipement l n'est allumé, alors l'équipement de refroidissement l est éteint.

$$\forall h, k, l \quad p_h \times c_{hl} \times b_{hk} \leq ce_{kl} \tag{4.17}$$

La contrainte 4.17 permet de définir que si au moins un hôte refroidi par l'équipement de refroidissement l est allumé, alors cet équipement est allumé.

À partir de ces équations, nous sommes capables de calculer la consommation du refroidissement d'un cluster k : C_k^{cool}. Pour cela, nous appellerons C_k^{ce} la consommation d'énergie des infrastructures de refroidissement du cluster k. La consommation d'énergie de l'ensemble des infrastructures de refroidissement du cluster k sera donc :

$$C_k^{cool} = C_k^{ce} \times \sum_{l=1}^{L} (ce_{kl}) \tag{4.18}$$

4.2.2 Objectif

Maintenant que l'ensemble de contraintes est bien défini, il reste à décider de la métrique que nous voulons utiliser pour définir l'objectif qui va décider de l'orientation de l'optimisation. Nous devons donc définir une fonction objectif pour chaque problèmes différents. Nous n'aurons évidemment pas le même objectif lorsque l'on voudra atteindre une certaine performance des tâches, et lorsque l'on voudra limiter la consommation électrique de l'infrastructure.

Performance

La première fonction objectif à définir nous demande de définir une métrique de performance. Pendant longtemps dans le domaine du calcul haute performance, la métrique dominante à été la performance. Le but du calcul haute performance était bien évidemment d'effectuer un calcul dans le moins de temps possible. Ainsi, il existe un bon nombre de métriques pour définir la performance d'une tâche, que ce soit des métriques de performance uniquement, comme le nombre de millions d'instructions par secondes (MIPS), ou des métriques de qualité de service, comme le temps de réponse.

Dans notre cas, nous choisirons une métrique définie par Stillwell et al. dans [101], appelée le *yield*, qui définit le pourcentage de *satisfaction* de la tâche. Le *yield*, est défnit par le ratio entre ce que la tâche avait demandé, et ce qui lui a été effectivement alloué. Ainsi, nous avons une métrique représentant la performance des tâches, relativement à ce qu'ils avaient demandé.

À partir de cette métrique, nous pouvons définir une fonction objectif pour nous permettre d'optimiser la performance. Nous définissons donc le yield minimum Y dans l'équation 4.19 :

$$\forall j, \quad \sum_h \frac{\alpha_{jh}}{r_{jk}} \geq Y \tag{4.19}$$

Cette fonction objectif nous permettra d'avoir à la fois la performance et l'équité, bien que privilégiant l'équité entre les tâches plutôt que leurs performances respectives. Ici, l'objectif sera de maximiser le yield mimimum, afin d'avoir le yield de la tâche la moins bien allouée le plus haut possible, et d'éviter d'avoir une majorité de tâches avec une allocation proche de ce qu'ils avaient requis, et une minorité avec une mauvaise allocation, ce qui se passerait si par exemple nous avions choisi d'utiliser le yield moyen. Nous pourrons fixer une limite sur la performance en définissant une borne sur le yield minimum, afin de traduire un SLA définissant une satisfaction minimale que les tâches doivent avoir. Cette limite sera définie par l'équation $Y >= Y^{bound}$, et décrira le problème que nous appellerons BOUNDEDYIELD.

Consommation d'énergie

La consommation d'énergie est aussi une métrique que nous devons avoir, que ce soit pour mesurer l'efficacité d'une allocation ou pour définir l'orientation de l'optimisation. Ainsi, nous définirons d'abord la consommation d'un hôte. Comme mentionné précédemment, nous définissons la consommation d'un hôte comme proportionnelle à la consommation de son CPU. En effet, la consommation d'une machine va varier en fonction de la charge de celle-ci, et l'augmentation de la charge va induire une augmentation de la vitesse du ventilateur. Par contre, les modules comme le disque dur, ou la mémoire en général ne seront pas impactés par la charge du CPU.

Ainsi, même si le modèle de consommation d'énergie n'est pas précis à 100%, nous pourrons avoir une estimation relativement précise de la consommation d'une machine, comme corroboré dans [91]. De ce fait, nous définissons cette consommation en 2 parties. Tout d'abord une partie statique, qui représentera la somme des consommation dont on ne peut se passer, comme par exemple l'alimentation, la consommation minimale du CPU, etc... Ensuite, nous aurons la partie dynamique, qui représentera tout le surcoût induit par la charge du processeur. Nous aurons donc la puissance consommée par un hôte représentée en l'équation 4.20 par sa partie statique, plus la partie dynamique en fonction de la charge CPU. On remarquera par ailleurs qu'avec ce modèle, un hôte éteint ne consommera rien.

$$E_h = C_h^{min} p_h + (C_h^{max} - C_h^{min}) \sum_j \alpha_{jh} \tag{4.20}$$

Nous pouvons maintenant définir la consommation en puissance d'un cloud par la somme des consommations des hôtes, en y ajoutant le refroidissement (équation 4.21).

$$E = \sum_h C_h^{min} p_h + \sum_h \left[(C_h^{max} - C_h^{min}) \sum_j \alpha_{jh} \right] + \sum_k C_k^{cool} \tag{4.21}$$

La consommation en puissance d'une infrastructure de cloud sera donc donnée par E. En regardant du côté de la fonction objectif, nous pouvons utiliser directement E comme métrique

à minimiser. Si nous parlons de réduire la consommation d'énergie d'un cloud, il faudra minimiser E. Si nous devons par contre fixer une limite sur cette énergie, cas que nous nommerons BoundedPower, nous n'aurons qu'à définir cette limite par $E = limite$.

Compromis

Le troisième problème discuté précédemment était celui du compromis. Un compromis mal géré peut amener à des situations mauvaises, lorsque par exemple on minimise E, si l'on gère mal le compromis on peut être amené à éteindre l'ensemble des hôtes, ce qui n'est pas une allocation valide. Que faire donc quand on n'a pas de borne sur aucune des deux métriques, et que l'on donne autant de valeur à l'énergie qu'à la performance ? C'est à cela que servira la métrique que nous appellerons Z et que nous devrons maximiser. Autrement dit, maximiser Z équivaudra à tenter de trouver le meilleur compromis entre l'augmentation de performance et la réduction de consommation d'énergie. Nous définirons cette métrique comme une combinaison linéaire entre la performance et l'énergie, c'est-à-dire optimiser $Z = \lambda Y + (1 - \lambda)(1/E)$, avec une valeur de λ comprise entre 0 et 1.

Cette définition répond bien au principe d'avoir une métrique permettant de favoriser plus ou moins la performance ou les économies d'énergie. Cependant, elle possède telle qu'elle des inconvénients majeurs.

Premièrement, le choix de la valeur de λ est problématique. Comme mentionné précédemment, comment choisir la valeur de λ ? Quelle valeur est pertinente ? Comment avoir le compromis "optimal" entre la performance et l'énergie et surtout qu'est ce que cela veut dire ? Ce problème est malheureusement complexe et nécessite une réflexion de fond sur la valeur d'une allocation. Comment décider si une allocation est bonne, ou plutôt comment choisir entre un système qui arrive à avoir 80% de performances pour 3kW de consommation de puissance, et entre un autre qui aura 75% de performance pour 2.8kW de puissance ? La réponse facile serait de dire que cela dépend de ce que le fournisseur de service veut, ou de ce que le client veut. Mais justement dans le cas où le fournisseur de services n'a pas de contraintes sévères ni sur la performance ni sur l'énergie, il est cohérent d'adopter une approche hybride.

De plus, bien que mettre une valeur pour λ proche de 1 définisse la priorité de l'allocation en faveur de la performance, le compromis résultant de l'allocation avec un tel λ n'est pas défini, puisque nous ne pourrons quantifier le montant de performance gagné par rapport aux économies d'énergie perdues.

Ensuite, l'utilisation d'une telle valeur n'offre aucune garantie quand au fait que la solution soit un optimum de Pareto. Cependant, nous pouvons corriger ce problème en calculant la solution au problème d'allocation borné en puissance (respectivement en performance) avec la valeur résultante de la puissance (respectivement du yield) de l'allocation de Z.

Enfin, nous nous retrouvons devant un problème qui concerne les ordres de grandeur des différentes composantes de Z. Nous avons d'une part le yield qui est compris entre 0 et 1, et une consommation de puissance d'une infrastructure, qui est elle théoriquement comprise entre 0 (toutes les machines sont éteintes) et E^{max} (toutes les machines sont allumées et consomment C^{max} de puissance). Il faudra donc normaliser E pour pouvoir avoir des ordres de grandeur comparables et ainsi avoir une plus grande facilité dans le choix de la valeur de compromis pour la combinaison linéaire Z. Une solution facile serait de diviser E par E^{max} afin d'être sûrs d'avoir une valeur comprise entre 0 et 1. Nous aurons donc la formule $Z = \lambda Y + (1-\lambda)(1-E/E^{max})$. Cette solution est simple et nous apporte ce que l'on veut, cependant elle possède des désavantages, comme par exemple le fait que la valeur minimale calculée par E/E^{max} ne sera en pratique pas 0, mais sera au moins égale à la somme des consommations induites par les tâches sur les hôtes les plus efficaces, divisé par E^{max}. De ce fait, puisque E/E^{max} n'est pas compris entre 0 et 1

en pratique, nous n'aurons pas le même effet de λ sur la composante yield, et sur la composante énergie.

Nous voyons bien ici que nous retombons au final sur un problème d'allocation, puisque pour trouver l'énergie minimale effective que nous pourrons avoir au sein d'une allocation, il faudra en trouver le résultat optimal, ce qui pose les mêmes problèmes de complexité. Nous pouvons par contre nous approcher un peu en tentant de calculer le nombre minimal d'hôtes que nous devrons en théorie utiliser, et par exemple utiliser le calcul de la borne rationnelle pour cela (cf : 3.5.3). Cependant, même en étant plus précis sur ces calculs, nous sommes toujours confrontés aux problèmes posés par un compromis entre de la performance et de l'énergie, à la différence d'impact et de signification des deux composantes, et au choix de la valeur de compromis.

Bien que ce problème possède des désavantages, il peut être intéressant de le calculer. Que ce soit parce que l'on n'a pas de préférence sur la performance et l'énergie, et que l'on tente au fil d'itérations successives d'arriver à une bonne valeur de compromis, ou que ce soit pour en extraire une borne pour re résoudre un problème borné en énergie ou performance ensuite, il restera utile dans certains cas de calculer des allocations dans le but de maximiser Z. Ce problème sera appelé MixedObjective par la suite.

4.3 Algorithmes

Afin de résoudre les différents problèmes, il faudra bien sûr adapter chaque type d'algorithmes pour effectuer des allocation cohérentes en fonction du problème. Ainsi, nous devrons définir les différentes orientations des algorithmes pour chaque problème.

Algorithmes gloutons

Nous définissons d'abord des algorithmes de type gloutons, c'est à dire des algorithmes qui calculent une allocation étape par étape, sans revenir sur leur choix afin d'avoir un optimum local.

Pour résoudre les différents problèmes, nous prendrons dans un premier temps le cas où nous n'avons qu'un seul cluster qui comporte des machines homogènes mais hétérogènes en énergie, et nous ne prendrons pas en compte le refroidissement.

Dans le cas du problème BoundedYield, nous définissons tout d'abord l'algorithme Greedy_BoundedYield_1 qui est un algorithme de type First Fit, qui allouera les tâches dans l'ordre des plus gros besoins mémoire (i.e : le tri des tâches se fait par m_j descendant). L'algorithme Greedy_BoundedYield_2 est aussi un algorithme First Fit, mais celui-ci triera les tâches par besoins CPU descendant. Les algorithmes que nous appellerons, Greedy_BoundedYield_3 et Greedy_BoundedYield_4 sont respectivement les versions best fit des algorithmes Greedy_BoundedYield_1 et Greedy_BoundedYield_2.

Pour calculer la solution au problème borné en puissance, nous définirons les algorithmes Greedy_BoundedPower_1 et Greedy_BoundedPower_2 qui fonctionneront de la même manière que les algorithmes de même numéro que pour le problème BoundedYield, au niveau des tris des tâches et des hôtes. Au départ, ces algorithmes tenteront d'allouer tous les tâches avec un yield $Y = 1$. Ensuite nous tenterons de réduire de manière itérative (par pas de 1%) les valeurs de α_{jh} (tant que celles-ci sont non nulles) de toutes les tâches dans un ordre suivant un *round robin*. Cette étape sera répétée jusqu'à ce que toutes les tâches puissent être allouées et que l'allocation arrive à une puissance consommée en deça de la borne. Bien évidemment, si une tâche se voit allouer un yield de 0 avant que ces conditions soient remplies, nous considérons donc que l'allocation échoue. De la même manière que pour le problème borné en performance, nous aurons les algorithmes Greedy_BoundedPower_3 et Greedy_BoundedPower_4 qui représentent

les versions Best Fit des algorithmes GREEDY_BOUNDEDPOWER_1 et GREEDY_BOUNDEDPOWER_2.

Enfin, pour le problème MIXEDOBJECTIVES, les algorithmes GREEDY_MIXEDOBJECTIVE_1 et GREEDY_MIXEDOBJECTIVE_2 qui vont être utilisés pour résoudre ce problème sont basés sur les algorithmes précédents qui solutionnent le problème BOUNDEDPOWER. Nous calculerons d'abord la valeur $H_Z = max(1, \lceil (1-\lambda) \times (\sum_j \alpha_j) \rceil)$, qui nous donnera une estimation théorique du nombre d'hôtes nécessaire pour allouer les tâches avec cette valeur de λ. La borne en énergie est ensuite calculée en utilisant les H_Z premiers hôtes (possédant les plus petit C^{max}). Nous utiliserons ensuite cette borne pour résoudre le problème BOUNDEDPOWER en utilisant un des algorithmes associés à ce problème. L'idée ici étant de calculer en premier le nombre théorique d'hôtes nécessaire pour allouer les tâches, afin de tenter de réduire la consommation d'énergie sur un nombre déjà réduit d'hôtes, et ainsi avoir plus de chance de trouver une allocation consommant en dessous de la borne.

Algorithmes de vector packing

Dans cette section, nous introduirons plus en détails la manière dont nous utilisons les techniques de vector packing afin de résoudre les différents problèmes.

Nous prendrons tout d'abord, pour le problème monocluster, l'algorithme de vector packing décrit précédemment, que nous allons modifier afin de tenter d'y intégrer des considérations de performances et d'énergie. Actuellement, cet algorithme tente d'allouer dans le moins d'hôtes possible les tâches en fonction de leur quantité de ressources demandées. Ainsi, cet algorithme ne prendra pas en compte dans ses calculs la consommation d'énergie des machines ciblées, et les économies qui peuvent résulter des allocations seront dues aux bonnes performances de l'algorithme. En effet, si celui-ci arrive à allouer sur seulement une moitié d'hôtes, l'autre moitié pourra être éteinte pour économiser de l'énergie. On pourrait donc voir la réduction de la consommation de puissance comme un effet collatéral de l'allocation via cet algorithme, et non comme un objectif à part entière.

Afin de réduire la consommation d'énergie via un tel algorithme, nous devons changer l'objectif actuel, qui est la quantité de ressources (ou le pourcentage par rapport à cette quantité), pour y intégrer des considérations à la fois de performance et d'énergie.

Nous introduisons donc une métrique que nous appellerons le *yield energy-aware* représenté par YE_{jh} qui nous servira de métrique objectif pour l'allocation de tâches dans l'algorithme EARESALLOC. Cette métrique que nous avons développé et étudié dans [24, 23] vise à palier à l'antagonisme entre la performance et l'énergie. YE_{jh} est défini comme suit :

$$YE_{jh} = \frac{(Y_{jh})^{1-\beta}}{(E_{jh})^{\beta}} = \frac{\left[\sum_{h=1}^{H}(\frac{\alpha_{jh}}{\alpha_j})\right]^{1-\beta}}{\left[\gamma\delta C_{jh} + (1-\gamma)(A_h(1-\sum_{j'=1,j'\neq j}^{J}(\alpha_{j'h})))\right]^{\beta}} \tag{4.22}$$

La partie Y_{jh} représente le yield de la tâche j, alors que la partie E_{jh} représentera la consommation en puissance résultante de l'allocation. Le paramètre β premet de définir le compromis entre la performance et l'énergie consommée. δC_{jh} représente le surcoût en puissance induit par la tâche sur l'hôte.

Nous définirons A_h comme "l'attractivité" d'un hôte, c'est-à-dire une valeur qui permet de déterminer si allouer une tâche sera plus efficace énergétiquement parlant sur une machine ou sur une autre. Le facteur $A_h(1-\sum_{j'=1,j'\neq j}^{J}(\alpha_{j'h}))$ nous servira à controler la consolidation plus ou moins importante des tâches sur les hôtes attractifs. Pour commencer, nous avons utilisé une valeur de $A_h = C^{max}$ pour privilégier les hôtes efficaces lorsqu'ils sont chargés.

Enfin, le paramètre γ visera à choisir et faire un compromis entre le placement (c'est-à-dire choisir l'hôte le plus efficace) et la consolidation (c'est-à-dire trouver un placement sur un nombre réduit de machines).

L'heuristique utilisant la métrique YE_{jh} sera donc un algorithme de vector packing, qui tentera de maximiser cette même métrique, en effectuant un placement avec l'algorithme décrit précédemment avec l'algorithme 3. Elle visera à réaliser un placement suivant 3 buts :

- maximiser la performance via le yield.
- placer les tâches sur les machines efficaces.
- consolider les tâches sur un nombre reduit d'hôtes afin d'éteindre ceux qui sont inutilisés.

Nous définissons donc un algorithme qui sera décliné en trois algorithmes visant à résoudre les trois différents problèmes.

Pour le problème BOUNDEDPOWER, nous effectuerons avec l'algorithme EARESALLOC_BOUND_E une recherche dichotomique sur le paramètre β, avec comme contrainte que la puissance résultante de l'allocation ne dépasse pas E^{bound}, la borne sur la puissance que le système ne doit pas dépasser.

Pour le problème BOUNDEDYIELD, nous allouerons les ressources avec l'algorithme EARESALLOC_BOUND_Y qui aura pour objectif une borne sur la performance Y^{bound} et une valeur de $\beta = 0$.

Enfin, le problème mixte MIXEDOBJECTIVE sera résolu par l'algorithme EARESALLOC_MIXED en utilisant la métrique YE_{jh} telle quelle, puisqu'elle a été construite de manière à résoudre cette approche bicritère. Nous prendrons donc une valeur de $\beta = \lambda$, faisant ainsi correspondre le paramètre de compromis λ de Z avec β.

Pour chaque problème, nous utiliserons une valeur de $\gamma = min(\frac{H}{J}, 1)$, ce qui veut dire que plus γ est proche de 1, plus nous tenterons de consolider, et inversement, plus γ sera proche de 0, plus nous tendrons à privilégier le choix des machines efficaces.

Solution optimale et borne sur l'optimal

Étant donné que les trois différents problèmes découlent du problème de l'allocation de tâche en tant que problème de *bin packing*, ceux-ci sont aussi des problèmes NP-complets, et ainsi leur formulation en tant que problème linéaire mixte entier et rationnels ne peut pas être résolue en temps polynomial. Nous ne pourrons donc calculer les solutions optimales que pour les petits problèmes, c'est-à-dire les problèmes avec un nombre réduit de tâches et d'hôtes.

Dans nos expérimentations, nous calculerons donc la solution optimale seulement pour les problèmes possédant moins de 6 hôtes et 16 tâches, celles-ci ont dans nos expériences eu un temps d'exécution de l'ordre de 3 minutes dans les pires cas. Nous appellerons dans ce qui suivra la résolution du programme linéaire de façon optimale MILP (*Mixed Integer Linear Programming*).

Cependant, puisqu'il n'est pas suffisant de calculer les résultats d'algorithmes sur des instances petites, dans la mesure où le système étudié est un système large échelle, nous voudrons aussi pouvoir comparer les algorithmes avec une borne haute. Ainsi, nous prendrons comme valeur de comparaison la borne rationnelle (c'est-à-dire en considérant que toutes les variables sont rationnelles), qui est une borne supérieure à la solution optimale. Cette solution s'appellera LPBOUND, et peut être calculée en temps polynomial.

Nous avons utilisé dans un premier temps le solveur open source GLPK[50] pour calculer MILP et LPBOUND. Il existe d'autres solveurs comme ILOG CPLEX[1], et Gurobi[54], qui nous permettent de calculer ces valeurs. Nous avons changé de solveur par la suite, pour passer de GLPK à Gurobi, d'une part pour sa performance, et d'autre part pour sa simplicité d'utilisation.

4.4 Validation expérimentale

4.4.1 Méthodologie monocluster

Afin de valider les modèles et algorithmes, ainsi que d'étudier les différents impacts et comportements du système, une validation et une étude expérimentale est nécessaire. Pour ce faire, nous avons développé un programme écrit en C++ de plus de 6000 lignes de code nous permettant de simuler une allocation selon différents algorithmes, systèmes et infrastructures. Ce programme nous permet simplement de définir en entrée l'état du système, c'est-à-dire définir les machines du cluster avec leur consommation C^{min}, C^{max}, spécifier les différentes tâches avec α_j, m_j, et enfin spécifier le problème et l'algorithme d'allocation. Nous récupérons en sortie l'allocation résultante, c'est à dire le mapping des tâches sur les hôtes, ainsi que leur valeur d'allocation (quel pourcentage ont été alloués pour les ressources fluides). De ces allocations, nous récupérons différentes statistiques sur le système, comme le yield minimum, moyen, la consommation d'énergie, le temps de calcul des algorithmes, etc.

Afin de conduire ces expérimentations, nous avons dû prendre des valeurs représentatives de certains aspects de systèmes réels, pour pouvoir ainsi avoir des valeurs en cohérence par rapport aux systèmes réels. Pour ce faire, nous avons récupéré des valeurs représentatives de C^{min} et C^{max} en récupérant des informations de consommation sur Grid5000, particulièrement sur le site de Lyon [78]. Nous avons choisi de prendre des valeurs de 120W de consommation de puissance lorsque les machines sont allumées, avec un écart-type de 20, et une valeur de 190W de consommation lorsque les machines sont chargées au maximum, avec un écart-type de 30. Ainsi, nous aurons des valeurs de C^{min} entre 100W et 140W, et des valeurs de C^{max} entre 160W et 220W. Ces valeurs seront calculées dans notre simulateur par une loi uniforme entre ces intervalles, le nombre des échantillons ne permettant pas de déterminer avec précision cette loi.

Dans les instances que nous avons générées, pour les différentes tâches, nous avons choisi, comme mentionné précédemment, de prendre uniquement deux types de ressources : une fluide qui est l'utilisation CPU, et une rigide qui est l'occupation mémoire. Ainsi, pour générer les différentes valeurs de la consommation CPU sur une machine référence (afin de pouvoir transformer par la suite cette valeur pour les différentes machines dans le cas de l'hétérogénéité), nous avons choisi de prendre une distribution normale avec une moyenne de 0.33 (cette valeur en particulier est expliquée par la suite), et un écart-type de 0.5. Bien évidemment, nous avons tronqué les valeurs inférieures ou égales à 0 (une tâche consomme des ressources CPU) et celles supérieures à 1 (une tâche ne peut consommer plus de ressources que ce qu'un hôte peut offrir). Ce faisant, nous avons dû ensuite artificiellement re-échelonner les valeurs pour arriver à la moyenne définie précédemment. Nous avons choisi ce modèle simple de génération de tâches, car nous voulions avoir d'une part un contrôle sur la part relative que l'ensemble des tâches demanderaient, et ensuite car il n'existait pas de modèle pour les besoins CPU exprimés en pourcentage d'hôtes, et qui permettait d'avoir une distribution suffisamment hétérogène des besoins CPU des hôtes. Cette approche est similaire à celle prise par Stillwell et al. dans [101].

Pour les besoins mémoire des tâches, nous avons choisi de prendre une distribution Pareto d'ordre 6, avec ceux-ci définis entre 2% et 95%, puisque dans beaucoup de systèmes, seulement une petite partie de tâches consommeront une large partie de besoins mémoire. C'est le cas lorsque l'on a par exemple une architecture de type 3 tiers de serveurs web, où nous aurons dans ce cas les tâches de type base de données qui consommeront plus de mémoire que les tâches de type serveur HTTP.

Bien que ce soit parfois le cas dans les systèmes réels, nous considèrerons le cas général où les besoins mémoire et CPU sont indépendants, c'est-à-dire que les tâches ayant de gros besoins CPU n'auront pas automatiquement des gros besoins mémoire, et vice-versa.

Nous avons ainsi des méthodes de génération d'hôtes et de tâches. Ce qui veut dire que nous sommes capables de générer un système. Il faudra maintenant choisir la taille du système. Comme mentionné précédemment, la taille de l'infrastructure étudiée est importante, dans la mesure où d'une part cela va conditionner la présence ou non de la solution optimale dans les résultats, et d'autre part que nous n'avons pas de garantie que les systèmes se comportent de la même manière avec des changements d'échelle. Nous devrons donc expérimenter sur plusieurs tailles de système, afin de voir si les algorithmes passent à l'échelle, et si cette échelle a une importance sur les résultats.

Pour ce faire, nous aurons à choisir deux paramètres : le nombre d'hôtes et le nombre de tâches. Le nombre d'hôtes et le nombre de tâches seront reliés, dans la mesure où puisque nous avons définis des distributions spécifiques pour la génération aléatoire des tâches, le nombre de tâches va agir sur la charge globale théorique du système. Dans notre cas, nous avons défini une génération de tâches avec une moyenne de 0.33, ce qui voudra dire que si nous prenons autant de tâches que d'hôtes, nous aurons un système chargé à 33% de sa capacité. Ainsi, il est intéressant de pouvoir expérimenter sur la charge relative des systèmes, et donc de voir l'impact sur les systèmes surchargés, ou sous-chargés.

Nous prendrons donc des valeurs pour le nombre de tâches égale à une, deux ou trois fois le nombre d'hôtes, ce qui équivaudra à une charge théorique de l'infrastructure de 33%, 66% et 99%. Nous aurons donc $J = \{H, 2H, 3H\}$, avec H faisant varier le système de petites instances à grandes instances, $H = \{4, 8, 16, 32, 64\}$. Il est intéressant de noter que dans les instances surchargées (à 99% et même moins), il est possible que l'on ne puisse pas trouver d'allocation valide, du fait de la forme des tâches, notamment de leurs besoins en ressource rigide.

Pour chacune de ces combinaisons (H, J), nous allons générer selon la méthode décrite précédemment 100 instances. Pour chacune de ces instances, nous allons résoudre le problème d'allocation pour les trois différents problèmes. Nous générons aussi aléatoirement les bornes contraignantes pour les problèmes qui en ont besoin, afin de pouvoir observer aussi l'effet des différentes bornes sur les différents algorithmes. Pour le problème BoundedYield, nous générons une borne qui sera exprimée en pourcentage du yield maximum théorique que l'on peut obtenir (c'est-à-dire sans prendre en compte l'allocation, juste les besoins par rapport aux capacités). Nous prendrons cette borne entre 50% et 100% de ce maximum théorique exprimé par $max(1, \frac{\sum_i \alpha_i}{H})$. De la même manière, pour le problème BoundedPower, nous prendrons un pourcentage du maximum de puissance consommable par le système, c'est-à-dire entre 50% et 100% de $\sum_h C^{max}$. Enfin, pour le problème mixte MixedObjective, puisque nous ne pouvons pas nous permettre de prendre à chaque résolution le spectre entier de valeurs de λ, nous prendrons 3 valeurs représentatives de λ qui seront 0.2 (allocation axée sur les économies d'énergie), 0.5 (allocation équilibrée) et 0.8 (allocation axée plutôt sur la performance).

Pour chaque résolution de problème, nous calculons un ensemble de valeurs nous permettant d'évaluer la performance relative de l'allocation. Ces valeurs comprendront au moins l'énergie du système E, le yield minimum Y, le temps de calcul. Nous ajouterons à cela le taux d'échec des algorithmes, qui représente le pourcentage de fois où l'algorithme a manqué de trouver une solution viable. Il faut noter que comme nous ne vérifions pas la faisabilité des instances générées, il sera possible d'avoir des cas où la solution optimale est un échec. Cependant bien sur, la grande majorité des instance a une solution puisque nous ne sommes pas dans des systèmes extrêmement contraints.

4.4.2 Résultats monocluster

Nous avons effectué les différentes expérimentations décrites en 4.4.1, et en avons tiré différents résultats que nous allons présenter dans cette section.

BoundedYield

Commençons tout d'abord par le problème d'optimisation de la consommation d'énergie sous contrainte de performance. Considérons tout d'abord les petites instances, puisque c'est pour celles-ci que nous aurons le plus d'informations, notamment la résolution optimale de chaque problème.

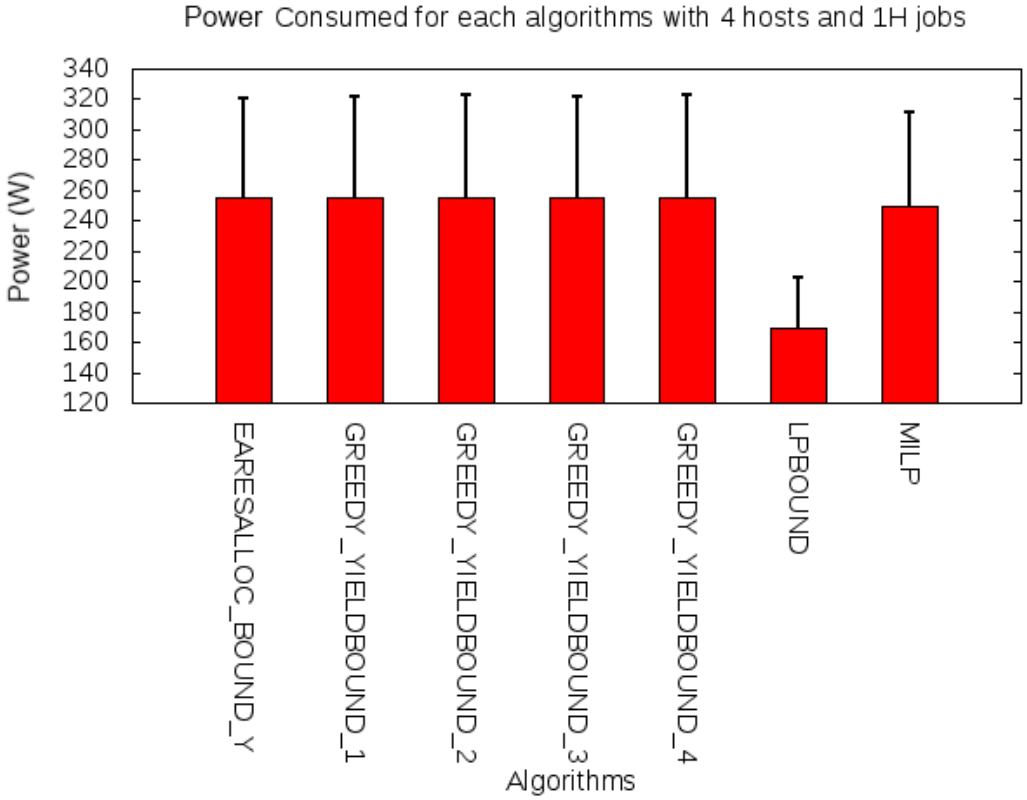


FIGURE 4.1 – Puissance moyenne et écart type consommés par les instances résultants de l'exécution des algorithmes pour 4 hôtes et 4 tâches

La figure 4.1 nous présente la puissance consommée moyenne (avec écart-type) pour chaque algorithme lorsque l'on possède 4 hôtes et 4 tâches. On peut voir ici ce qui avait été mentionné précédemment, c'est-à-dire le fait que tous les algorithmes ont des performances similaires en moyenne, avec un écart type plus petit pour l'algorithme MILP (qui est l'optimal), qui sera forcément plus consistant et meilleur que les autres. On remarque aussi que la valeur calculée par LPBOUND excède la performance de la valeur optimale, puisqu'elle nous donne toujours une allocation inatteignable, mais nous sert de borne supérieure sur la valeur optimale.

En concordance avec la figure 4.1, la figure 4.2 nous montre le nombre moyen d'hôtes utilisés par les différents algorithmes lorsque nous avons $H = 4$. Seulement environ 40% des hôtes sont utilisés, avec plus de 50% de réduction de puissance (comparé au cas sans consolidation des tâches). Evidemment, dans ce scénario, le système est loin d'être surchargé, et effectuer des économies d'énergie est facile. Néanmoins, ces résultats nous montrent l'importance d'avoir des algorithmes prenant en compte la puissance consommée. De plus, même des algorithmes simples

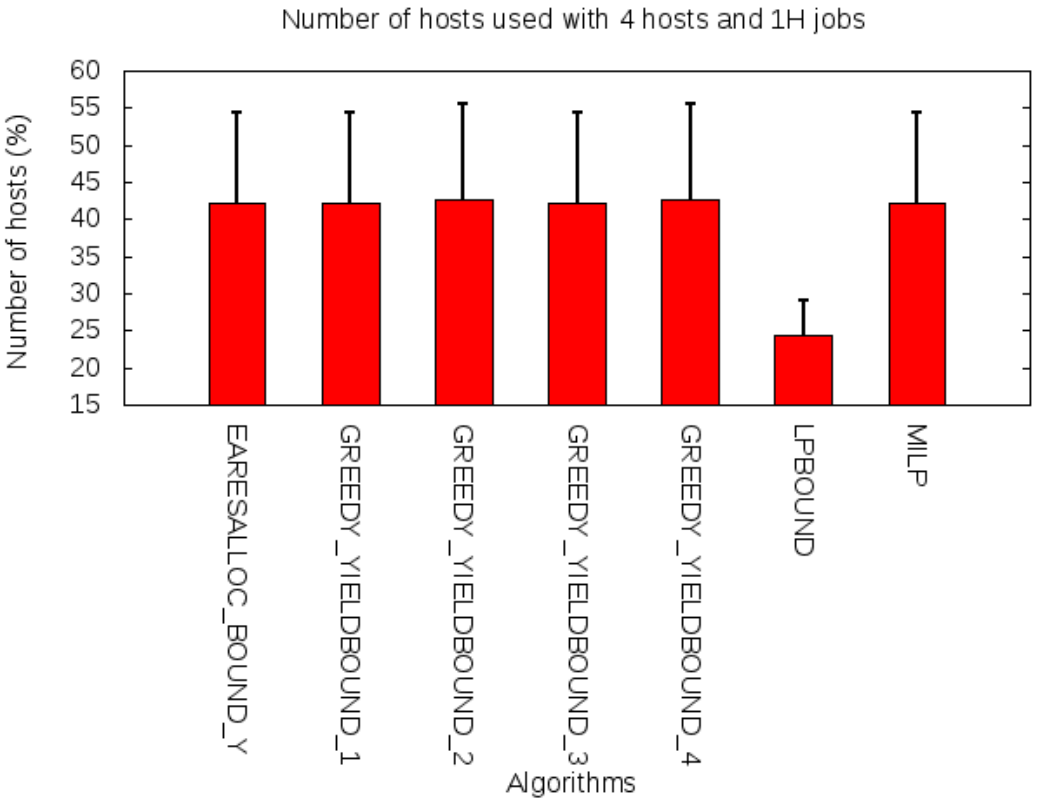


FIGURE 4.2 – Nombre d'hôtes moyens utilisés pour les instances de 4 tâches et 4 hôtes

TABLE 4.2 – Energy consumed for each algorithms

Algorithme	**Puissance (Watts)**	**Distance à la borne**
LPBOUND	1908	0%
EARESALLOC_BOUND_Y	2090	+9.5%
GREEDY_YIELDBOUND_1	2199	+15.2%
GREEDY_YIELDBOUND_2	2145	+12.4%
GREEDY_YIELDBOUND_3	2203	+15.4%
GREEDY_YIELDBOUND_4	2143	+12.3%

peuvent dans certains cas aboutir à des économies d'énergie significatives. Par ailleurs, de la même manière que pour la figure 4.1, nous voyons que les résultats donnés par LPBOUND sont inférieurs à une utilisation de 25% des hôtes, ce qui veut dire que dans certains cas, l'allocation résultante n'utilisera qu'une fraction d'une seule machine.

L'aggrégation des résultats de toutes les instances nous donne le tableau 4.2. Nous pouvons y voir la consommation d'énergie moyenne calculée sur s'ensemble des instances, ainsi que la le pourcentage de performance relative comparé à LPBOUND. Nous voyons ici que l'algorithme EARESALLOC_BOUND_Y est le plus performant, bien que ce soit de peu.

Afin de pouvoir y voir plus clair dans les effets en jeu dans ce cas là, la figure 4.3 nous

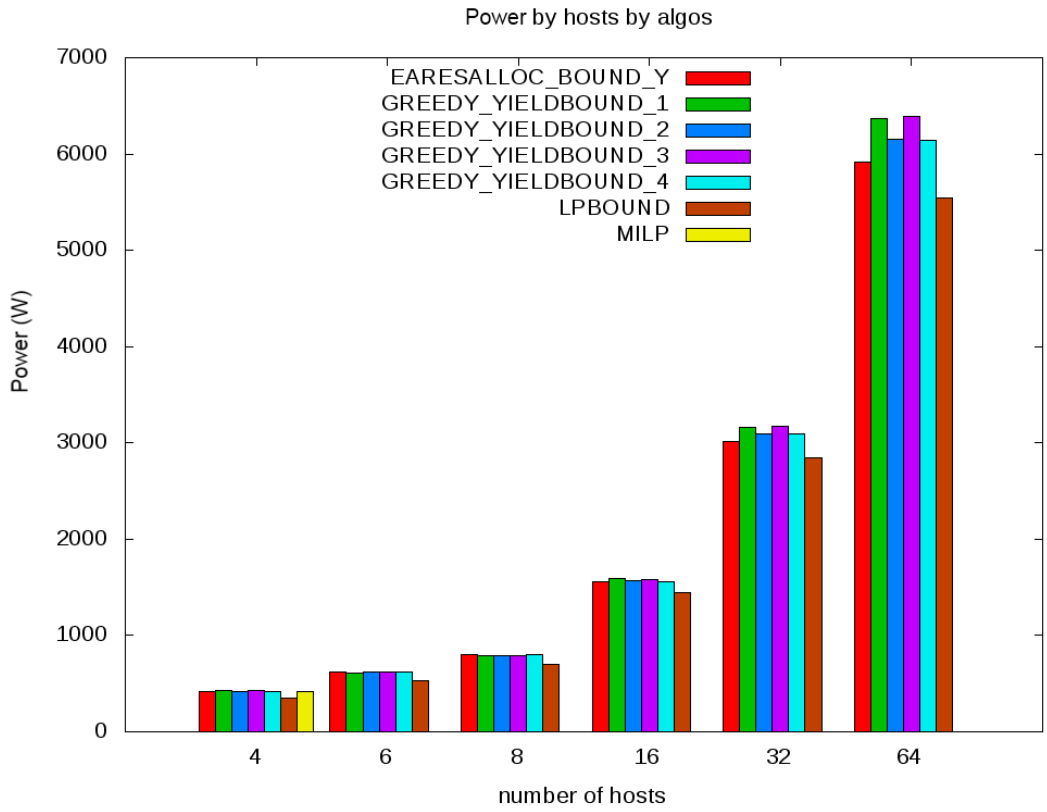


FIGURE 4.3 – Energie consommée par les instances résultantes de chaques algorithmes, pour chaque taille d'instance. L'optimal (MILP) n'est calculé que pour des instances de 4 hôtes

présente l'énergie moyenne consommée par chaque algorithme, regroupée par nombre d'hôtes en jeu dans les instances. Nous n'aurons les résultats pour MILP seulement pour les instances où $H = 4$, pour des raisons de temps de calcul expliquées précédemment. Comme on pouvait s'y attendre, les plus faibles économies d'énergies sont données par MILP et LPBOUND. Pour des petites instances (i.e : 8 hôtes et moins), toutes les heuristiques nous donnent des résultats similaires, comme nous l'avons vu précédemment pour 4 hôtes. Cependant, un écart se creuse lorsque les instances sont avec $H = 16$ et plus avec EARESALLOC_BOUND_Y pour atteindre le plus gros écart dans les plus grosses instances ($H = 64$). Dans de telles instances, nous aurons une consommation de puissance moyenne de $5900W$. De plus, EARESALLOC_BOUND_Y se trouve en moyenne à seulement 6% de LPBOUND, alors que le meilleur des algorithmes gloutons, GREEDY_BOUNDEDYIELD_4, se trouve à un écart de 10%. Au vu des écarts donc entre le moins bon et le meilleur des algorithmes, ainsi qu'entre le meilleur des algorithmes et la borne rationnelle sur l'optimale, nous pouvons donc conclure que les algorithmes proposés donnent dans l'absolu de bons résultats.

Nous pouvons remarquer que les algorithmes GREEDY_BOUNDEDYIELD_2 et GREEDY_BOUNDEDYIELD_4 ont des résultats meilleurs que GREEDY_BOUNDEDYIELD_1 et GREEDY_BOUNDEDYIELD_3, car comme mentionné précédemment, le tri des tâches qui permet l'allocation des tâches les plus grosses en premier donnent majoritairement des meilleures performances.

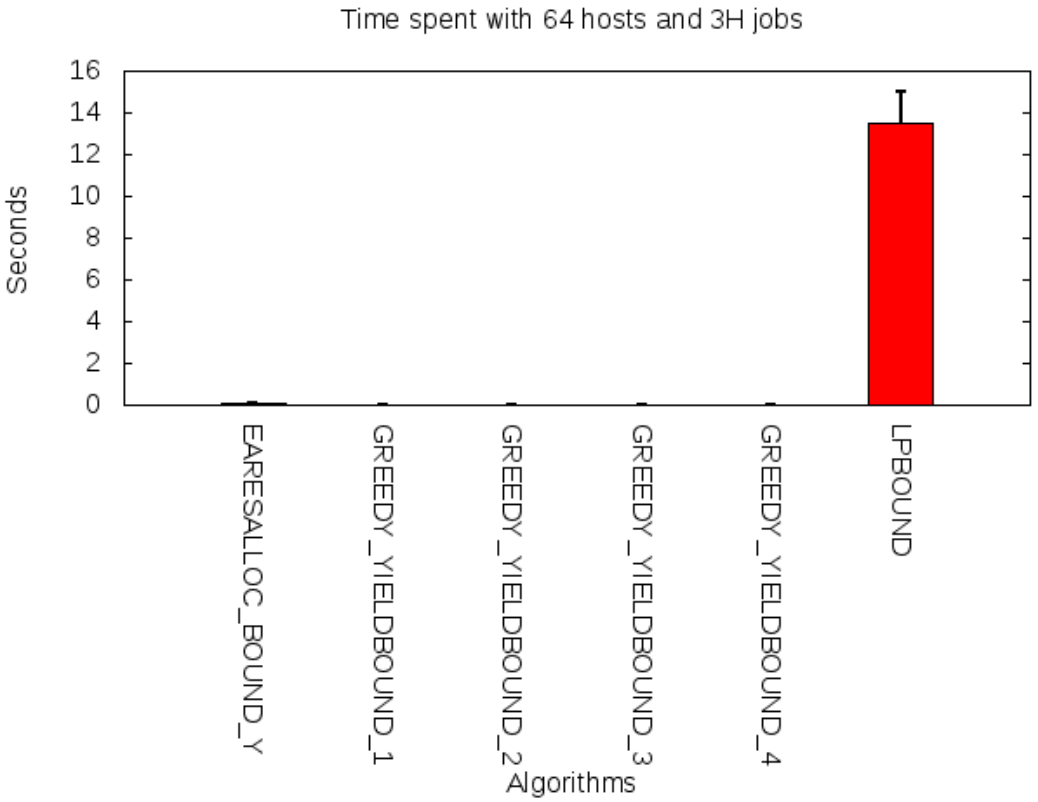


FIGURE 4.4 – Temps d'exécution moyen des différents algorithmes pour des instances de 64 hôtes et 192 tâches

Il est important de préciser que dans la figure 4.3, les instances avec tous les nombres de tâches sont regroupées, c'est à dire qu'il n'y a pas de distinction entre $J = H$, $J = 2H$ et $J = 3H$. On pourrait alors se demander si les résultats pourraient différer entre les différents ratios hôtes/tâches. En examinant ces mêmes résultats pour les différents nombres de tâches, on s'aperçoit que comme on pouvait s'y attendre, les algorithmes ont des meilleures performances pour les systèmes peu contraints ($J = H$), et tendent à se rapprocher de la borne rationnelle lorsque l'on augmente le ratio tâches/hôtes, et dans ces cas là, il y a moins de marge de manoeuvre pour les économies d'énergie, puisque rappelons le, lorsque l'on a $J = H$ (respectivement $J = 2H$ et $J = 3H$), cela représente une charge CPU de 33% du système en moyenne (respectivement 66% et 99%) en prenant juste en compte les besoins des tâches.

Cependant, l'ordre de performance des différents algorithmes n'est pas altéré quelque soit le nombre de tâches que l'on prend par rapport au nombre d'hôtes. Les résultats donnés en figure 4.3 restent donc valides.

Il nous reste maintenant, pour le problème BOUNDEDYIELD, à évaluer la performance des algorithmes en matière d'utilisabilité. C'est-à-dire évaluer d'une part le temps de calcul des algorithmes, puisqu'on ne peut pas utiliser un algorithme qui prend trop de temps à trouver une solution, et d'autre part le pourcentage d'échec à trouver une solution, car on ne peut pas non plus utiliser un algorithme qui échoue trop souvent et est n'est donc pas fiable.

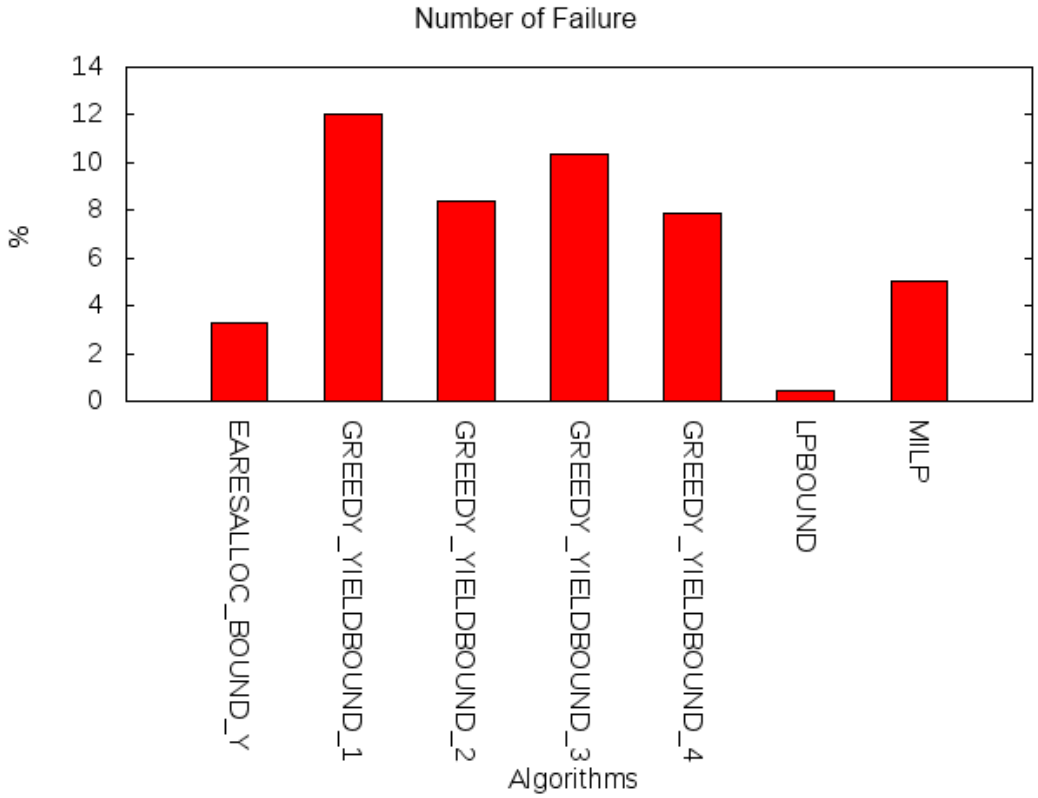


FIGURE 4.5 – Pourcentage d'échec des algorithmes

Le temps de calcul pris par les différents algorithmes est affiché en figure 4.4, pour les plus grosses instances (i.e : $H = 64$ et $J = 3H = 192$). On peut y voir que le temps de calcul est très bas, à part pour les solutions données par LPBOUND qui prennent en moyenne plus de 12 secondes. Les temps de calculs vont de 67 millisecondes en moyenne pour EARESALLOC_BOUND_Y à 1 milliseconde pour les algorithmes gloutons. Nous avons poussé les expériences pour EARESALLOC_BOUND_Y jusqu'à des instances de 500 hôtes et 1500 tâches, et avons obtenu des solutions en moyenne aux alentours d'une seconde (ce qui correspond au temps de résolution moyenne de la solution optimale MILP avec 4 machines et 12 tâches). Par conséquent, les algorithmes pour résoudre ce problème sont suffisamment rapides et scalables pour être utilisés. Les algorithmes gloutons ont une complexité en $nlog(n)$ alors que les algorithmes de vector packing ont une complexité quadratique.

Le pourcentage moyen d'échec des algorithmes est donné en figure 4.5. Nous pouvons y voir que l'algorithme EARESALLOC_BOUND_Y échoue dans un peu plus que 3% des cas, alors que l'algorithme glouton qui échoue le plus (GREEDY_BOUNDEDYIELD_1) a un taux d'échec d'environs 12%. En comparaison, l'algorithme glouton qui échoue le moins souvent est l'algorithme GREEDY_BOUNDEDYIELD_4, et possède un taux d'échec d'environ 8%. Comme on pourrait le deviner, la majorité de ces échecs vient des instances où $J = 3H$, puisque ce sont les systèmes les plus constraints (sans compter que certaines ne seront même pas faisables). On peut voir par ailleurs que MILP semble échouer plus que EARESALLOC_BOUND_Y, la raison

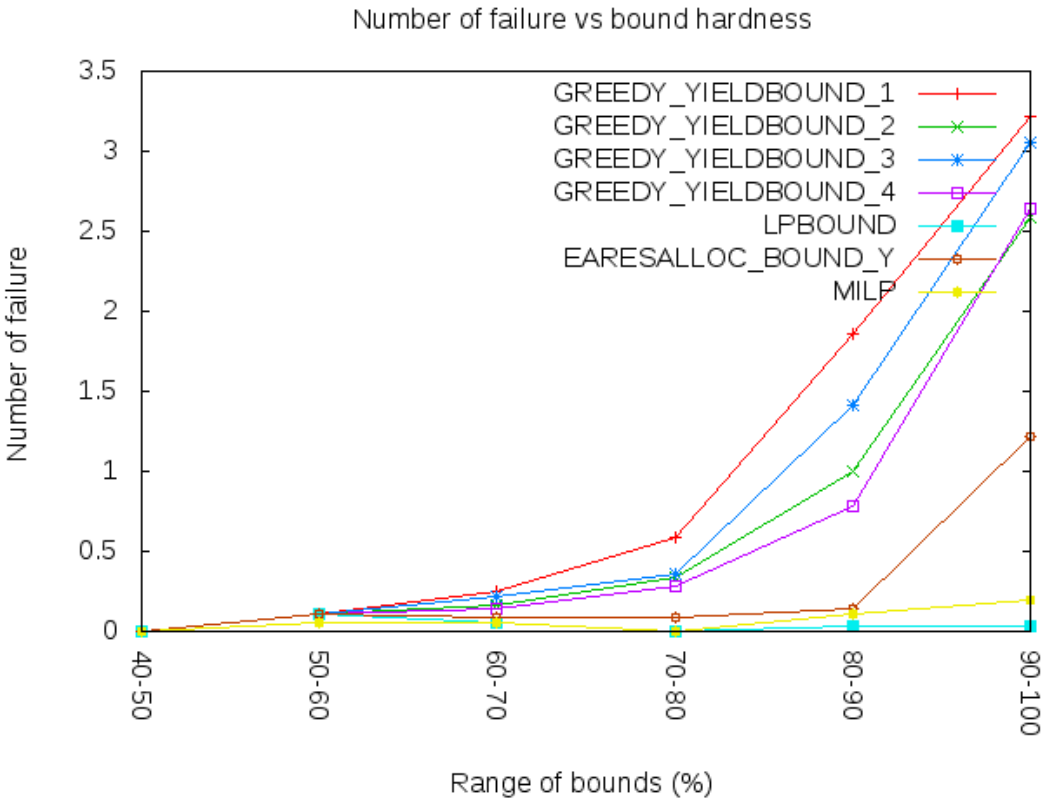


FIGURE 4.6 – Pourcentage d'échec en fonction de la dureté de la borne

étant que MILP est uniquement calculé sur les instances avec 4 hôtes.

Une fois que nous avons pu regarder la performance des algorithmes au regard du taux d'échec, il est important de noter que, puisque nous générons aléatoirement les bornes, certaines seront plus dures que d'autres. Pour étudier ce phénomène, la figure 4.6 nous montre le pourcentage d'échec en fonction de ce que nous appellerons la "dureté de la borne", qui définit le pourcentage relatif entre la borne du yield qui a été spécifiée, et la borne maximale qui peut être atteinte, qui sera dans le cas de ce problème 1.0. Cela veut dire par exemple qu'une dureté de borne de 90% représente une allocation où l'on doit atteindre au moins 90% du maximum théorique atteignable. Comme mentionné précédemment, nous ne générons que des valeurs de bornes entre 50% et 100%, d'où les valeurs d'abscisse de la figure 4.6. Chaque point représenté sur cette figure représente la somme des valeurs à l'intérieur de l'intervalle correspondante, et la somme de toutes les intervalles nous donne les valeurs représentées dans la figure 4.5.

Nous pouvons voir sur cette figure que, comme on pouvait s'y attendre, plus la dureté de la borne est importante, plus le taux d'échec des algorithmes est important. On y voir que pour les bornes entre 90% et 100% du maximum atteignable, les algorithmes gloutons échouent tous pour environ 3% de la totalité des instances (ce qui amène que 25% des instances échouées par l'algorithme GREEDY_BOUNDEDYIELD_1 le sont entre 90% et 100%). Nous pouvons voir ici que l'algorithme EARESALLOC_BOUND_Y est plus robuste que les algorithmes gloutons, et que le gros de la différence de résultats se fait justement lorsque la borne est dure.

BoundedPower

Maintenant que nous avons étudié les résultats pour le problème du BoundedYield, il faut faire de même pour le problème BoundedPower. Notons que pour ce problème, il y a une différence fondamentale avec le problème précédent. Pour le problème BoundedPower, la borne sur la consommation de puissance va décider du nombre d'hôtes maximum qu'il sera possible d'utiliser pour l'allocation.

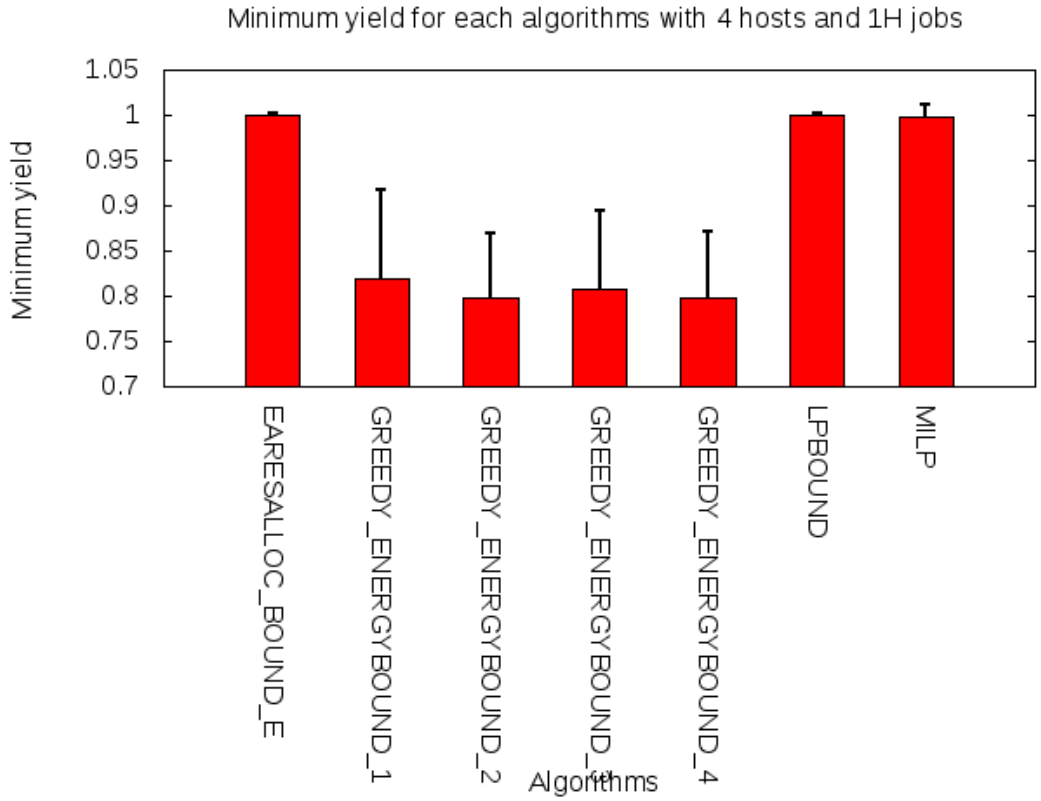


Figure 4.7 – Mean minimum yield for 4 hosts and 4 tâches

La figure 4.7 nous montre la performance des algorithmes pour 4 hôtes et 4 tâches. Contrairement à ce que nous avions vu pour le problème BoundedYield, les différences entre les algorithmes sont bien marquées. Cela est dû en majeure partie au fait que les algorithmes gloutons font la résolution de ce problème en deux étapes : faire une allocation, puis à partir de cette allocation tenter de réduire l'énergie en dessous de la borne. Ce qui se passe résulte donc de ce constat, et cette manière de faire va résulter pour les algorithmes gloutons à de fortes réductions dans le yield minimum. Ainsi, ces algorithmes vont avoir des performances allant aussi bas qu'une distance de 20% en moyenne des solutions optimales. En contraste, l'algorithme EaResAlloc_Bound_E montre des performances correctes, bien qu'en dessous de l'optimal sur certaines instances. La raison principale étant que l'heuristique de EaResAlloc_Bound_E tente de placer efficacement les tâches, et à la fois de les consolider, et ainsi, sur de telles instances, le résultat est parfois une consolidation supérieure au placement, donnant donc des résultats nous empêchant de se rapprocher de l'optimal.

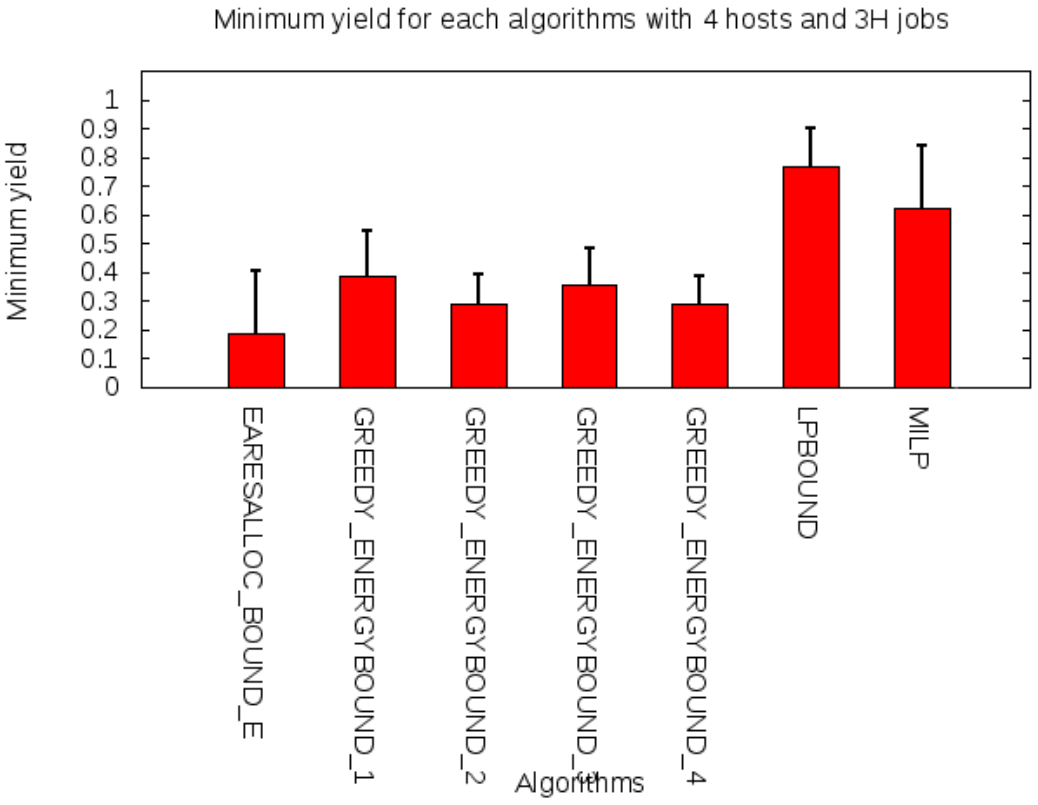


FIGURE 4.8 – Mean minimum yield for 4 hosts and 12 tâches

L'augmentation du ratio tâches/hôtes tend à diminuer plus encore la performance des algorithmes gloutons, comme on peut le voir en figure 4.8, où nous avons $J = 3H$. Dans ce cas là, la performance diminue jusqu'à 40% de la solution optimale. Cette augmentation du ratio diminue aussi les performances de l'algorithme EARESALLOC_BOUND_E, puisqu'il n'y aura souvent que peu voire pas de place pour consolider les tâches, et cela fera donc diminuer fortement le yield minimum. Notons aussi que cet algorithme possède le plus grand écart type, ce qui veut dire qu'il y a une plus grande variabilité des résultats, à cause du nombre d'échecs des allocations.

Intéressons-nous maintenant aux résultats regroupant l'ensemble des instances, comme le montre la figure 4.10. Les meilleurs résultats sont évidemment pour MILP et LPBOUND. Nous pouvons voir que l'algorithme EARESALLOC_BOUND_E donne en moyenne les meilleurs résultats, c'est-à-dire le meilleur yield minimum, parmi les heuristiques.

Bien que celui-ci ait montré des mauvaises performances dans les scénarios avec peu d'hôtes et un nombre relatif de tâches plus grand (cf : figure 4.8), dans des instances moins contraintes en ressources, nous aurons des performances plutôt bonnes.

Pour mieux comprendre ce qui se passe et comprendre la différence entre EARESALLOC_BOUND_E et les algorithmes gloutons, prenons la figure 4.9 qui nous montre le nombre d'hôtes utilisés pour des instances avec $H = 64$ machines et $J = 2H = 128$ tâches. Nous y voyons que EARESALLOC_BOUND_E utilise en moyenne bien plus d'hôtes que les algorithmes gloutons. Ce qui s'explique par d'une part par le fait que EARESALLOC_BOUND_E tente d'avoir une bonne

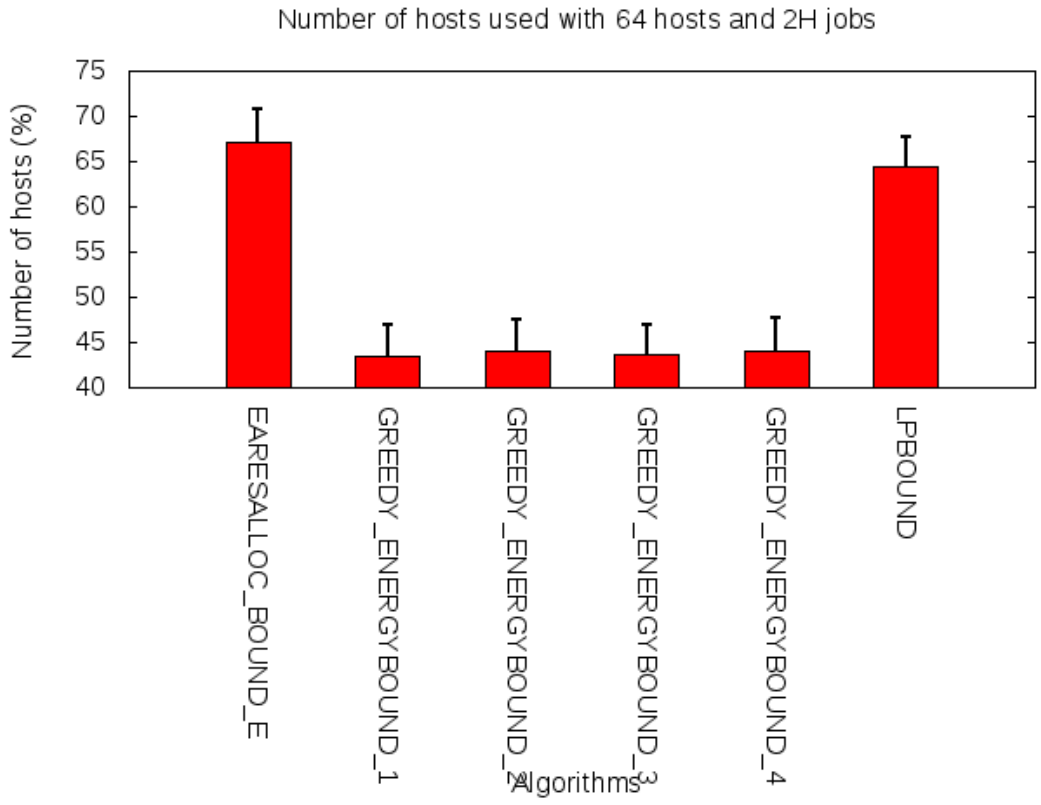


FIGURE 4.9 – Number of hosts used for 64 hosts and 128 tâches

consolidation mais aussi un minimum yield assez haut, ce qui amène un plus grand nombre d'hôtes, et d'autres part parce que les algorithmes gloutons favorisent la consolidation, ce qui se fait souvent au détriment du yield des tâches, mais donne un nombre d'hôtes plus faible.

Comme pour le problème précédent, intéressons nous maintenant au temps de calcul et au taux d'échec des algorithmes pour le problème BOUNDEDYIELD. La figure 4.13 nous montre le temps de calcul des algorithmes pour 64 hotes et 192 tâches. Comme nous pouvons le voir, l'algorithme EARESALLOC_BOUND_E prend beaucoup plus de temps que les algorithmes gloutons. Cela est dû au fait que cet algorithme effectue plusieurs allocations en tentant de trouver la meilleure allocation efficace en énergie (deux recherches dichotomiques sont en jeu). Ainsi, calculer une solution pour des instances de 500 hôtes et 1500 tâches prend un peu plus d'une minute.

La figure 4.11 nous montre la valeur du taux d'échec des algorithmes. Comme nous pouvions nous en douter avec le grand écart type de EARESALLOC_BOUND_E, celui-ci possède le plus gros taux d'échec, aux environs de 13%. La raison étant que l'algorithme ne favorise pas assez la consolidation des tâches, menant l'allocation à des échecs dans les cas où les instances ont une forte charge de tâches.

Comme précédemment pour la figure 4.6, la figure 4.12 représente le taux d'échec des algorithmes, regroupé par intervalles de dureté de borne. Contrairement à la figure 4.6, les valeurs basses représentent les bornes les plus contraignantes, puisque les valeurs générées sont en pour-

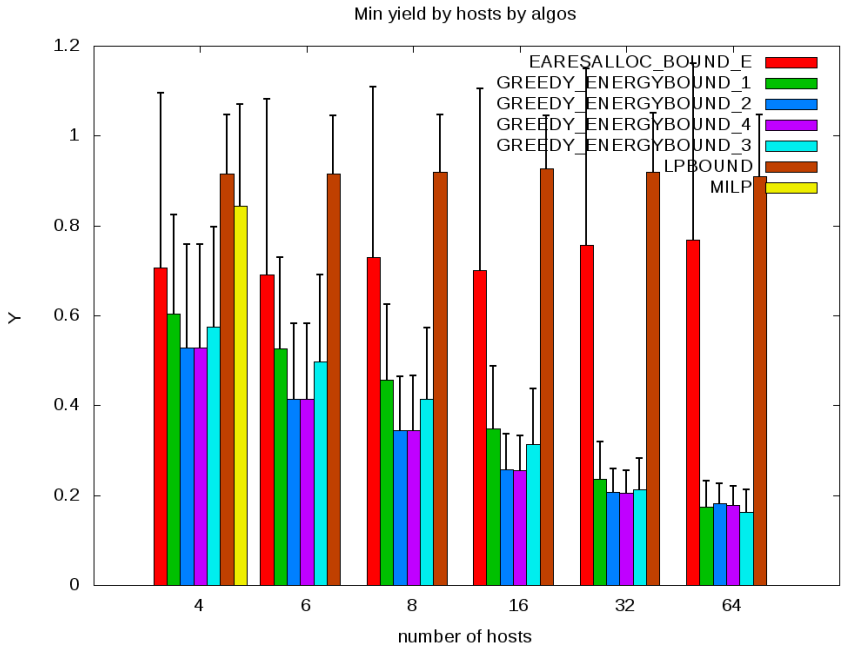


FIGURE 4.10 – Yield minimum donné par les algorithmes en fonction du nombre d'hôtes

centage de la consommation d'énergie maximale ($\sum_h(C_h^{max})$). Par exemple, une borne de 60% voudra dire que le système ne doit pas dépasser $0.6 \times \sum_h(C_h^{max})$, ce qui sera difficile à faire, surtout dans les scénarios fortement chargés en demande de ressources.

Nous pouvons voir sur la figure 4.12 que tous les algorithmes échouent à peu près aussi souvent pour des bornes entre 50% et 70%, et que l'algorithme EARESALLOC_BOUND_E échoue deux fois plus pour ces valeurs. Cependant, si le taux d'échec diminue de manière significative pour les bornes supérieures à 70%, l'algorithme EARESALLOC_BOUND_E possède toujours un taux d'échec non négligeable. La plupart de ces échecs sont dûs, comme mentionné précédemment, au déséquilibre de la consolidation des tâches effectué par cet algorithme pour les instances avec $J = 3H$.

MIXEDOBJECTIVE

Le dernier problème est le plus épineux à étudier. Le problème MIXEDOBJECTIVE tente de diminuer la consommation d'énergie tout en augmentant la puissance de calcul, ou plus exactement, tenter de diminuer la consommation d'énergie sans trop diminuer la performance. Le coeur du problème tient dans ce "sans trop", puisque nous avons un problème de quantification multi objectifs. De plus, les différents algorithmes n'optimisent pas la même métrique (par exemple, EARESALLOC_MIXED optimise sa propre métrique, alors que LPBOUND et MILP optimisent Z).

Comme nous allons le voir par la suite, il est tout de même intéressant de calculer ces solutions, et de voir comment se comportent les algorithmes dans les différentes situations, en fonction notamment des différentes valeurs de λ.

La figure 4.14 nous présente les résultats des algorithmes pour $\lambda = 0.20$, qui correspond au cas où nous favorisons la réduction de la consommation d'énergie par rapport à la performance

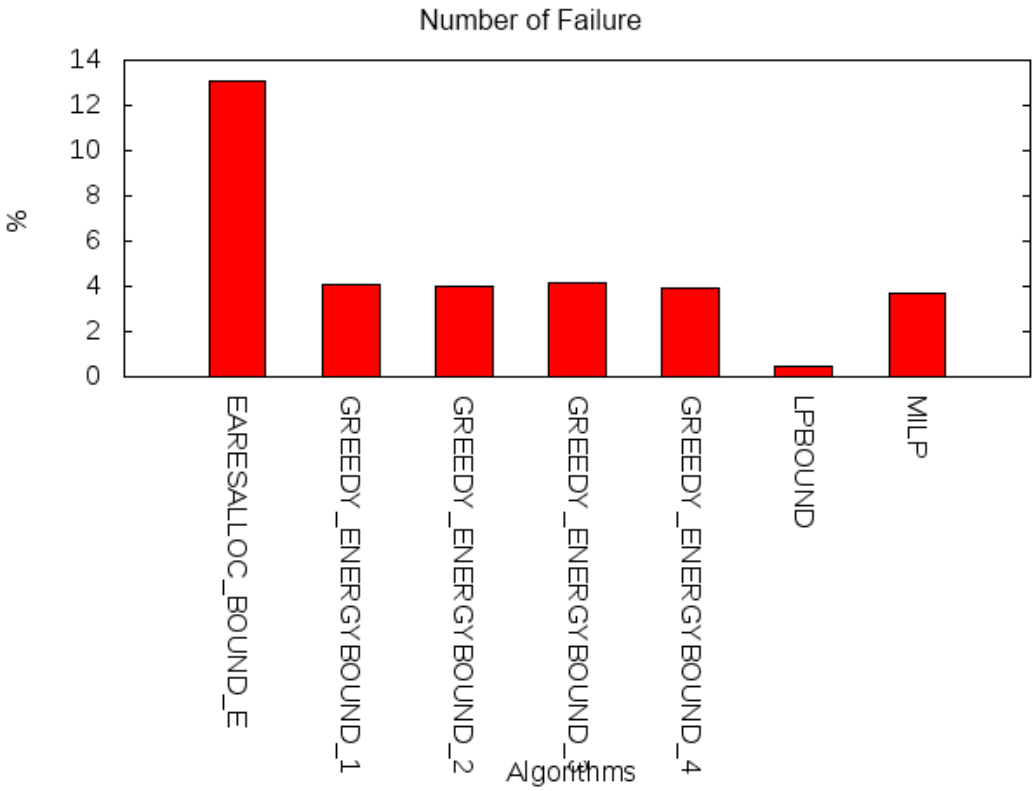


FIGURE 4.11 – Taux d'échec des algorithmes

des services. Sur cette figure les différents groupes de points, qui correspondent aux différents nombres d'hôtes, et qui font des "paliers" de consommation d'énergie entre les différents groupes de points. On notera que les groupes de points à un yield proche de 1 correspondent à des instances où nous avons $J = H$, alors que les groupes de points proche d'un yield de 0.4 correspondent à des instances où l'on a $J = 2H$.

On notera que multiplier le nombre d'hôtes par 2 ne veut pas nécessairement dire multiplier la consommation d'énergie par 2 pour les différents algorithmes. Nous pouvons voir ici que d'une part l'algorithme EARESALLOC_MIXED donne en général une plus grande consommation d'énergie que l'algorithme GREEDY_MIXED_1 par exemple, mais aussi d'autre part que le minimum yield est aussi meilleur. Etant donné que nous voulons, en choisissant $\lambda = 0.20$ favoriser le minimum yield, une plus importante consommation d'énergie importera moins, puisque nous serons plus proche de notre but.

Le cas où nous n'avons aucune préférence entre la performance et la consommation d'énergie est montré en figure 4.15, qui représenté le cas où $\lambda = 0.5$. Dans ce cas là, nous pouvons voir que l'algorithme EARESALLOC_MIXED a vu ses performances diminuer par rapport à l'expérience précédente. La plus raison la plus probable étant que, comme mentionné précédemment, la valeur de λ n'a pas le même impact sur le yield minimum que sur la consommation d'énergie. Il en résultera ici que cette valeur de λ n'a pas un effet suffisant sur la composante d'énergie. Autrement dit, l'impact de λ sur le yield minimum n'est pas proportionnel à celui sur la consommation

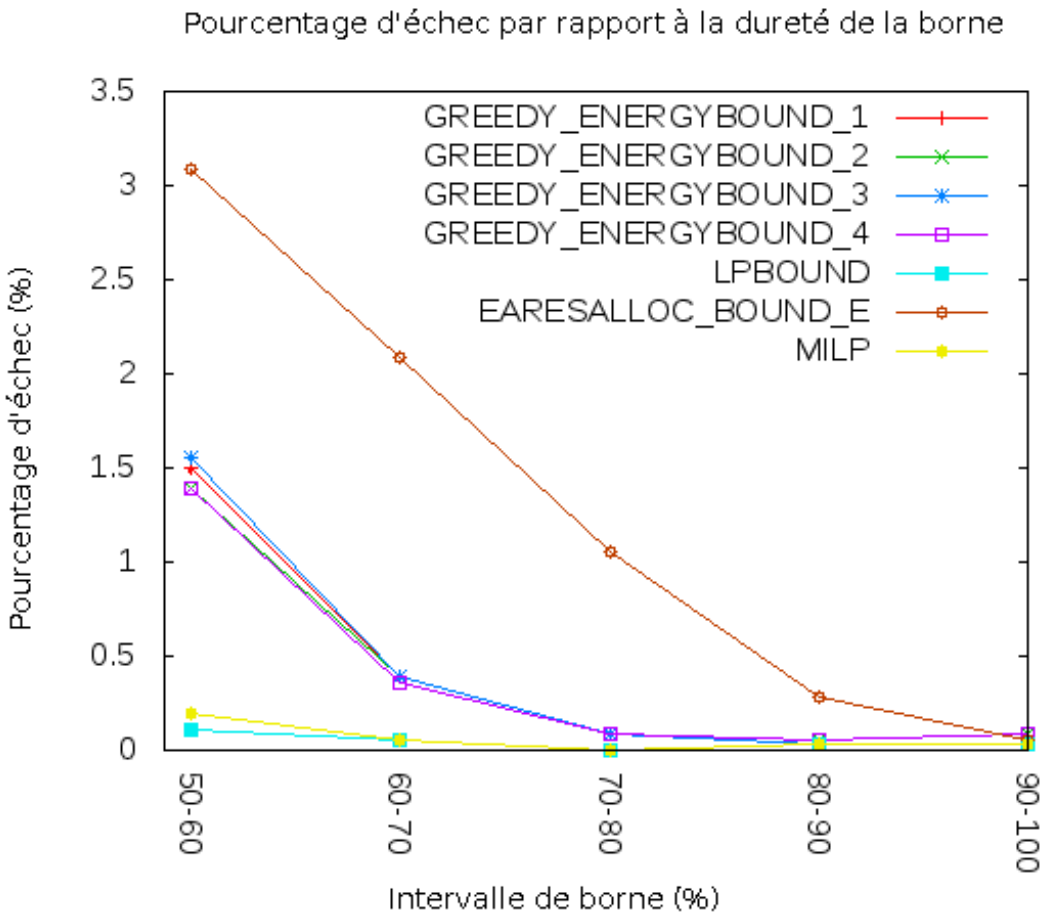


FIGURE 4.12 – Taux d'échec total des algorithmes en fonction de la dureté de la borne

d'énergie. Ainsi, l'impact sur le yield va avoir de l'importance alors que celui sur l'énergie n'en aura que peu, et par conséquent nous aurons un yield diminué avec une énergie seulement peu réduite.

Ce résultat nous donne l'intuition qu'il peut y avoir un (ou plusieurs) point(s) d'inflexion(s) autour duquel diminuer une partie de la métrique se fait avec peu d'effets sur l'autre partie de cette même métrique.

La figure 4.16 nous montre le cas où $\lambda = 0.8$. Dans cette expérience nous voulons réduire la consommation d'énergie, quitte à perdre fortement en performance. En effet, en regardant les résultats, la perte en performance est grande, à tel point que la plupart des valeurs de yield minimum sont en dessous de 0.5. Par contre, contrairement aux deux expériences précédentes, les diminutions en terme de consommation de puissance sont aussi significatives.

Au final, le paramètre β (qui sert de valeur de compromis dans la métrique utilisée par EARESALLOC) de la métrique n'est pas relié de la même manière aux différentes composantes de la métrique que le paramètre λ. C'est pourquoi ces résultats ne montrent pas les comportements auxquels nous aurions pu nous attendre en fonction des valeurs de λ. Pour atteindre les économies

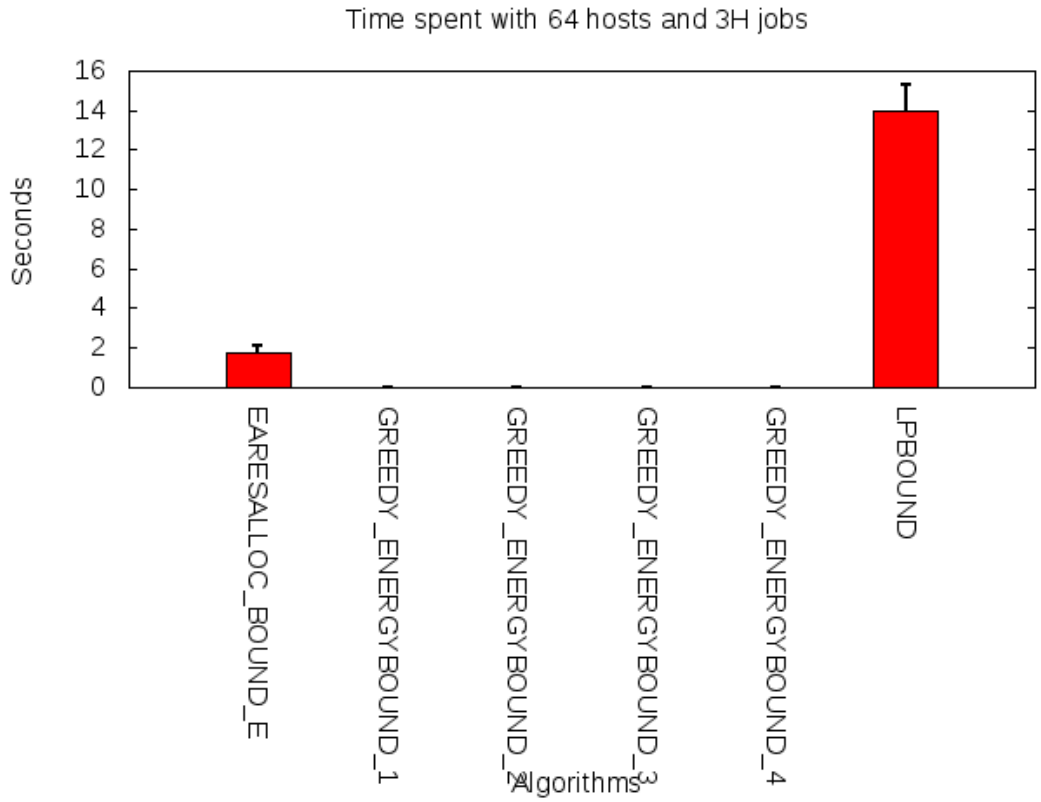


FIGURE 4.13 – Temps d'exécution des algorithmes pour des instances de 60=4 machines et 192 tâches

d'énergies escomptées avec $\lambda = 0.8$, il est possible qu'il faille par exemple choisir une valeur de $\beta = 0.9$, la relation entre les deux paramètres n'étant pas définie.

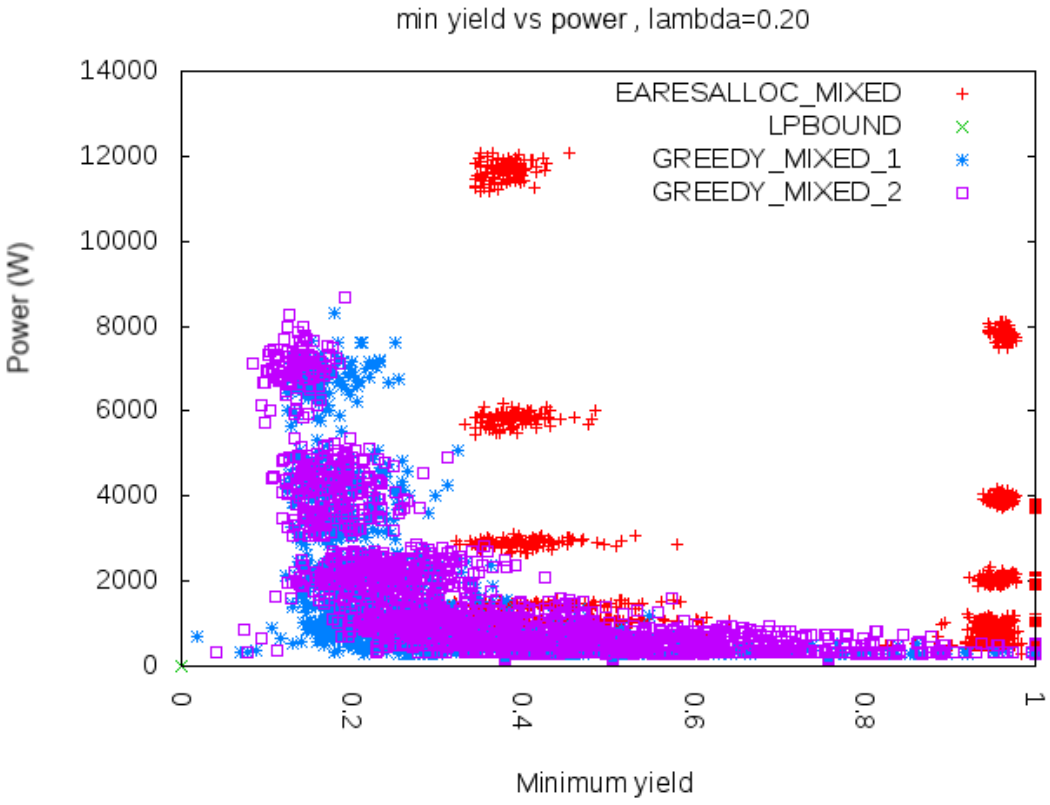


FIGURE 4.14 – Yield minimum en fonction de l'énergie consommée pour $\lambda = 0.2$, le cas où la consommation d'énergie est favorisée

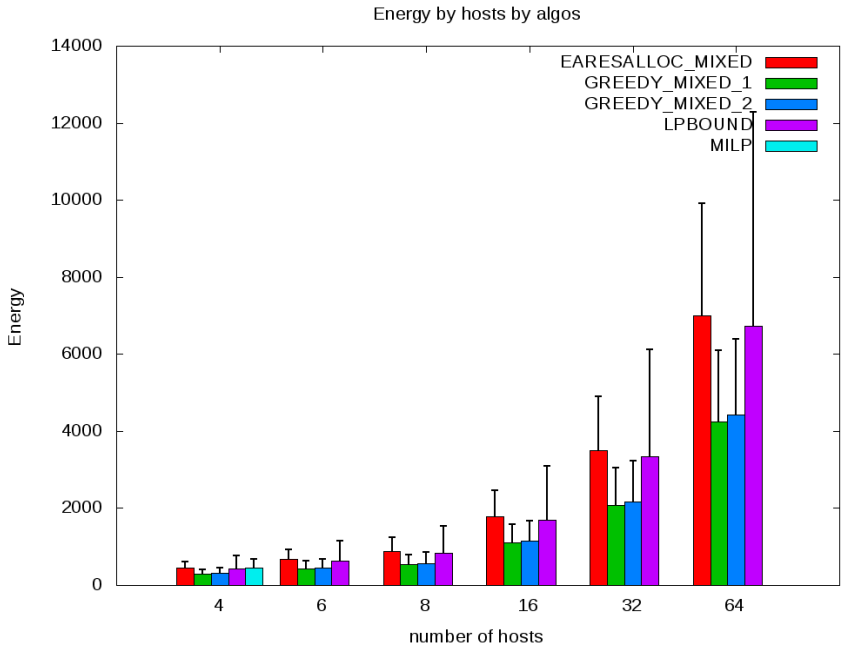


FIGURE 4.17 – Consommation d'énergie résultante de l'allocation par les différents algorithmes pour chaque taille d'instance

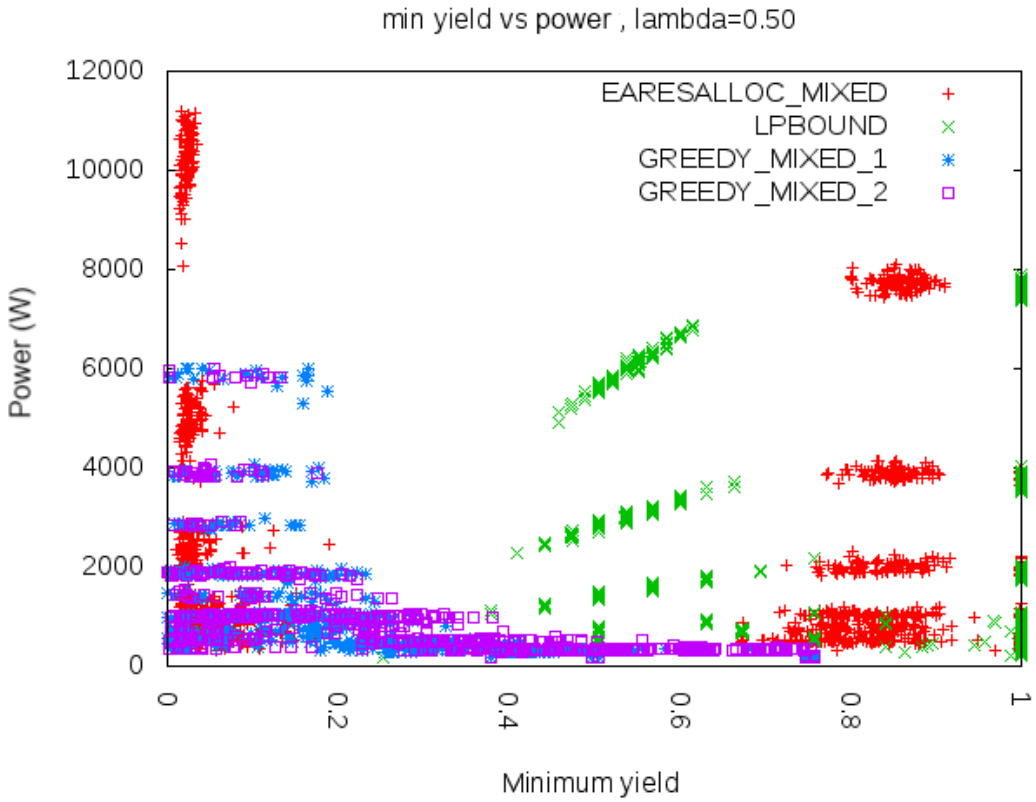


FIGURE 4.15 – Yield minimum en fonction de l'énergie consommée pour $\lambda = 0.5$, le cas où performance et énergie sont équilibrés

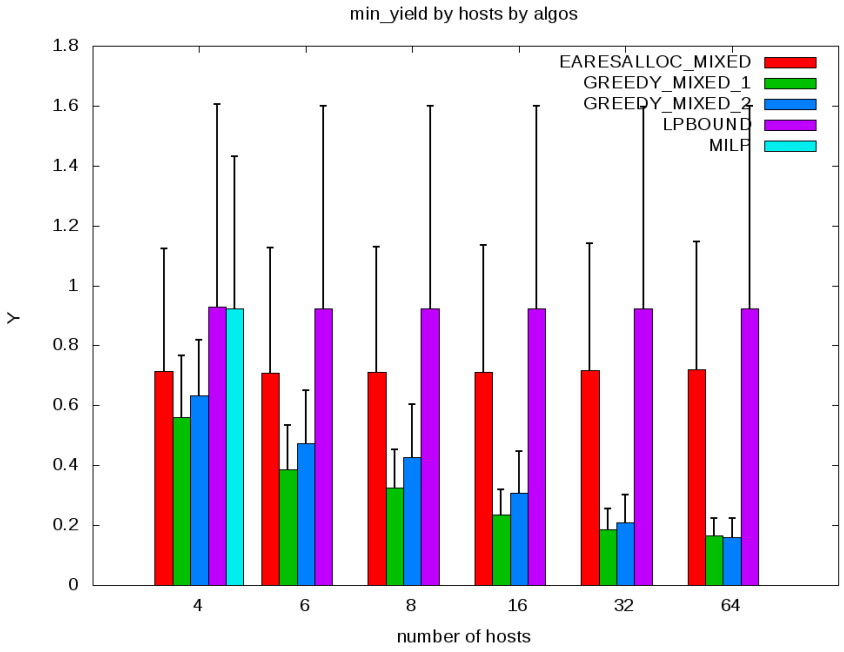


FIGURE 4.18 – Yield minimum résultant de l'allocation par les différents algorithmes pour chaque taille d'instance

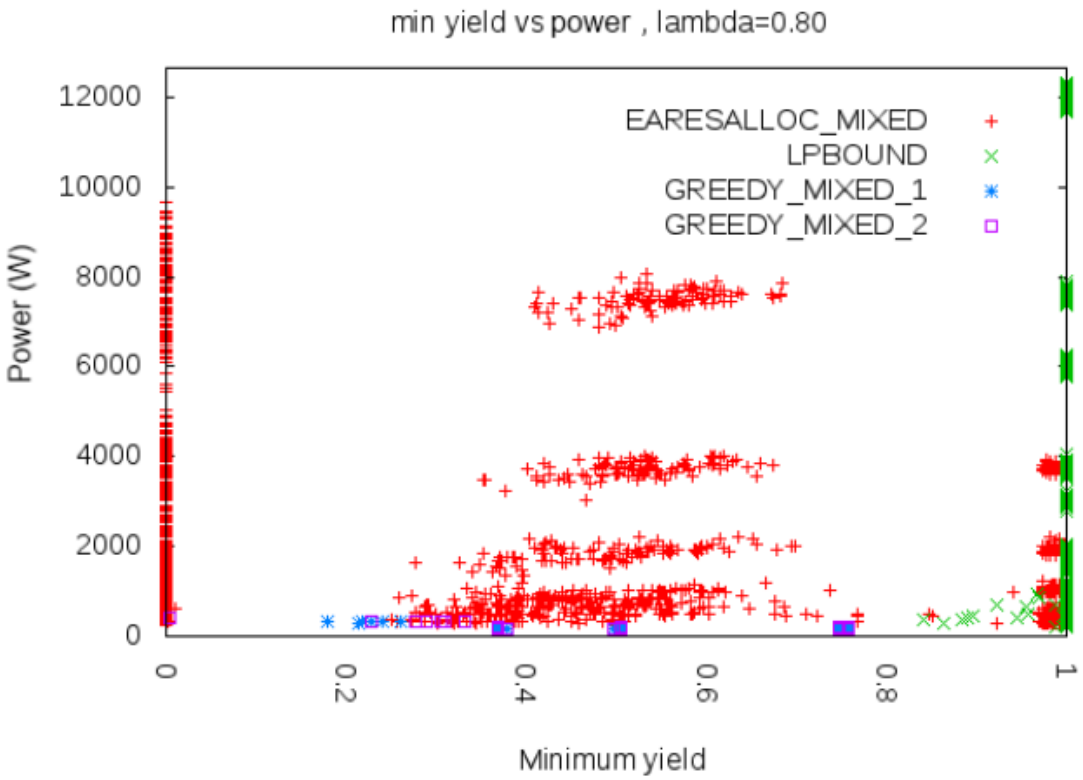


FIGURE 4.16 – Yield minimum en fonction de l'énergie consommée pour $\lambda = 0.8$, le cas où la performance est favorisée

Les figures 4.17 et 4.18 représentent respectivement la puissance moyenne consommée et le minimum yield moyen pour chaque algorithme, regroupée en fonction du nombre d'hôtes des instances. Nous pouvons y voir que l'algorithme GREEDY_MIXED_1, qui est plutôt focalisé sur les économies d'énergie, nous donne en moyenne les meilleures consommation d'énergie. En contrepartie, ce même algorithme donnera les pires résultats en terme de yield minimum.

L'algorithme EARESALLOC_MIXED fait quant à lui au contraire la plus grande consommation de puissance en moyenne pour le meilleur yield minimum, démontrant ainsi de la difficulté de comparer des résultats d'allocation bi-critères.

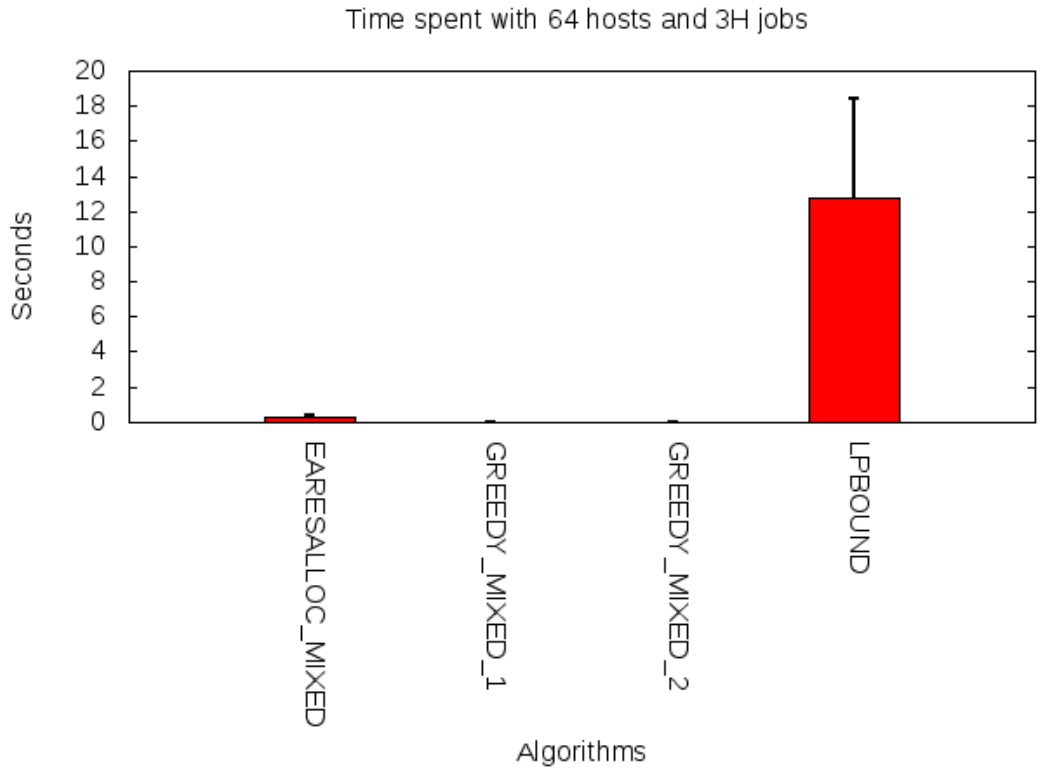


FIGURE 4.19 – Temps d'exécution des algorithmes pour des instances de 64 machines et 192 tâches

La figure 4.19 nous montre le temps d'exécutions des algorithmes pour $H = 64$ hôtes et $J = 3H = 192$ tâches. Nous pouvons y voir ici que l'algorithme EARESALLOC_MIXED calcule les solutions en moyenne en moins d'une seconde. Evidemment, cela est toujours moins rapide que les algorithmes gloutons qui calculent une solution avec un temps de l'ordre de plusieurs millisecondes, mais cela reste largement acceptable.

4.4.3 Passage au multicluster

Les expériences précédentes nous ont montré d'une part l'importance du choix de l'algorithme dans la résolution des problèmes, mais aussi que certains algorithmes, même simples, peuvent donner des résultats très bons, notamment dans les problèmes contraints en puissance ou en performance.

Seulement ces résultats ne prennent en compte qu'un cluster unique composé d'hôtes homogènes. Pour passer à un cloud multi clusters il faudra donc changer le fait que les hôtes soient homogènes, pour introduire le fait que les hôtes soient différents, et expérimenter sur ces nouveaux paramètres.

Prenons tout d'abord la méthodologie d'expérimentation. Puisque le problème devient plus complexe à mesure que l'on rajoute des paramètres à l'intérieur, nous avons choisi de simplifier les algorithmes. Les expériences précédentes ont montré l'utilité d'avoir une métrique et un

algorithme personnalisé et efficace pour résoudre les différents problèmes. Nous nous baserons donc sur cet algorithme, mais nous enlèverons les interactions trop complexes liées au multi objectif, c'est-à-dire $\gamma = 1$ pour l'algorithme EARESALLOC.

Rappelons ensuite que nous avions utilisé pour la génération des services monocluster une machine de référence, afin de pouvoir définir les besoins sur une autre machine en fonction. Par exemple, si un service demande 50% de CPU sur un serveur de référence à 1 GHz, alors il demandera 25% de CPU sur un serveur à 2 GHz.

Puisque nous envisageons de rajouter ensuite l'impact du refroidissement, il est intéressant de ne garder que les choses dont nous connaissons le fonctionnement. C'est pourquoi nous avons choisi d'utiliser des algorithmes gloutons qui fonctionnent de manière différente (Round Robin et First Fit), non pas pour leurs performances respectives, mais pour leurs fonctionnements antagonistes qui servira de base à notre algorithme efficace, qui lui sera basé sur la même technique de vector packing que EARESALLOC.

Nous n'utiliserons donc par la suite que ces trois algorithmes avec différentes variations sur les tris, ou sur la manière dont on va allouer les tâches, pour étudier les effets de chaque type d'allocation sur les différents systèmes.

Nous avons bien sûr modifié chacun pour prendre en compte ce qui nous intéresse. Pour l'algorithme FIRSTFIT, pour le problème BOUNDEDYIELD, l'allocation se fait avec des tâches qui ont un yield égal à la borne. Afin d'améliorer les performances de cette allocation, les tâches avec les plus hauts besoins rigides sont allouées en premier, sur les hôtes avec les plus petits C^{max}. Pour le problème BOUNDEDPOWER, l'algorithme fonctionne de la même manière, mais en effectuant une recherche dichotomique sur le plus haut minimum yield atteignable.

Pour l'algorithme ROUNDROBIN et le problème BOUNDEDYIELD, l'allocation se fait normalement, à un yield égal à la borne. Pour le problème BOUNDEDPOWER, l'allocation se fait aussi à la manière du round robin (une tâche par hôte de manière circulaire), mais en réduisant le yield du système résultant de l'allocation précédente si celle-ci est considérée comme surchargée, ou réduisant le nombre d'hôtes disponibles pour l'allocation si elle ne l'est pas, le tout jusqu'à atteindre la borne de puissance.

Comme mentionné précédemment, l'algorithme VECTORPACKING (VP) fonctionnera comme décrit pour EARESALLOC. Les solutions optimales et bornes sur l'optimal MILP et LPBOUND seront calculées de la même manière que précédemment, si ce n'est que le solveur utilisé ne sera plus GLPK mais Gurobi.

Pour la méthodologie de génération des instances, nous avons gardé la même, sauf que les hôtes ont des consommation allant de 50W à 80W pour C^{min}, et de 100W à 200W pour C^{max}, ces paramètres de génération étant basés sur des chiffres disponibles sur le site de specPOWER [34]. Les hôtes de chaque cluster sont générés pour avoir des capacités entre 1.5 GHz et 4 GHz pour le CPU, et entre 4Go et 16Go pour la RAM, selon . Les tâches changent aussi pour suivre une distribution normale allant jusqu'à 2000 MIPS, ce qui correspond à 40% du plus gros hôte. Enfin, après avoir vérifié les performances des différents algorithmes, nous ne prendrons des résultats que sur des grosses instances, ici 400 machines pour 1000 hôtes.

La première expérience vise à évaluer les performances des algorithmes décrits précédemment dans le cas où nous avons 6 hôtes et 16 tâches, afin d'avoir les valeurs optimales. C'est ce que montrent les figures 4.20 et 4.21. Comme on pouvait s'y attendre en regardant la figure 4.20, l'algorithme ROUNDROBIN possède les pires performances en termes d'énergie consommée, puisque celui-ci tente de disperser les tâches autant que possible. Cependant, comme nous pouvons le voir sur la figure 4.21, cela n'amène pas de meilleurs résultats en terme de yield minimum, puisque cette tendance à allouer les tâches sur le plus grand nombre d'hôtes peut amener à ne pas être capable de satisfaire la contrainte en puissance, et ainsi mener à des diminutions en performances.

Nous pouvons aussi voir sur ces figures que l'algorithme VECTORPACKING est celui qui pos-

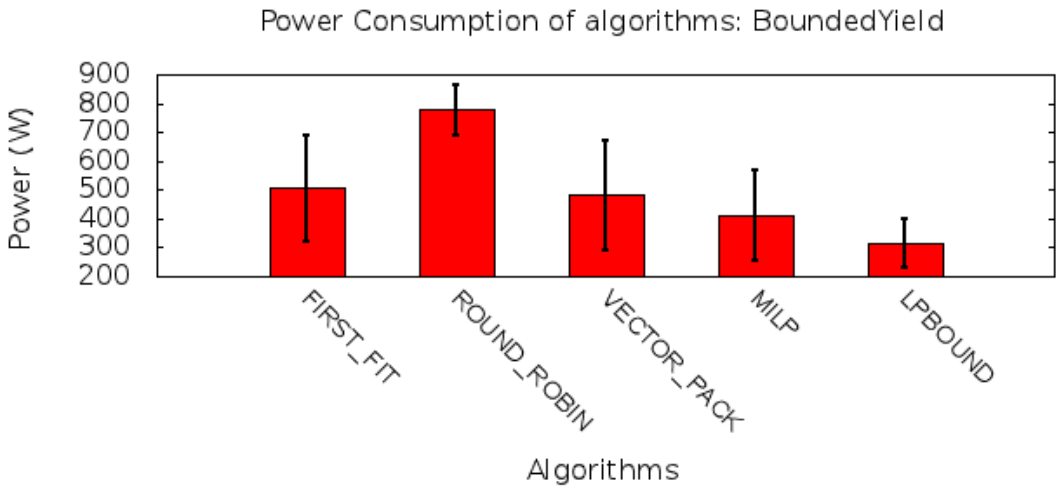


FIGURE 4.20 – Consommation de puissance résultant des allocations calculées par les différents algorithmes

Problème	Algorithme	Moyenne	Valeur mesurée
BoundedPower	VP	0.851953	Minimum Yield
	VP_GLOBAL_H	0.914844	
BoundedYield	VP	26887	Puissance électrique
	VP_GLOBAL_H	25876	

TABLE 4.3 – Résultats pour l'expérience sur l'hétérogénéité

sède les meilleures performances, en étant à une distance de 15% de la borne optimale au niveau de la puissance. Cet algorithme étant le plus performant, il est celui que nous choisirons pour les expériences suivantes avec différents paramètres et stratégies d'allocation.

Introduisons maintenant l'hétérogénéité dans les expériences et étudions son impact sur le yield minimum pour le problème BOUNDEDPOWER et sur la puissance consommée pour le problème BOUNDEDYIELD. L'algorithme VP modifié pour y introduire des considérations d'hétérogénéité telle qu'un tri en fonction des différentes capacités de ressources sera nommé VP_GLOBAL_H. Celui-ci tente d'allouer les tâches sur les hôtes qui ont le plus haut ratio capacité en ressource d'un hôte/C^{max}. Comme nous pouvons le voir dans la table 4.3, prendre en compte la nature hétérogène de l'infrastructure de manière simple peut améliorer le minimum yield jusqu'à 7% en moyenne. En regardant du côté de la puissance, la version hétérogène de VP nous donne environs 4% en moyenne d'économies d'énergies. Dans les deux cas, ces modifications viennent aussi avec une baisse dans l'écart type des résultats, ce qui veut dire une augmentation de consistance dans les résultats.

Par la suite, nous ne regarderons plus les résultats de FIRSTFIT et ROUNDROBIN et nous nous concentrerons seulement sur l'algorithme VECTORPACK et ses variations.

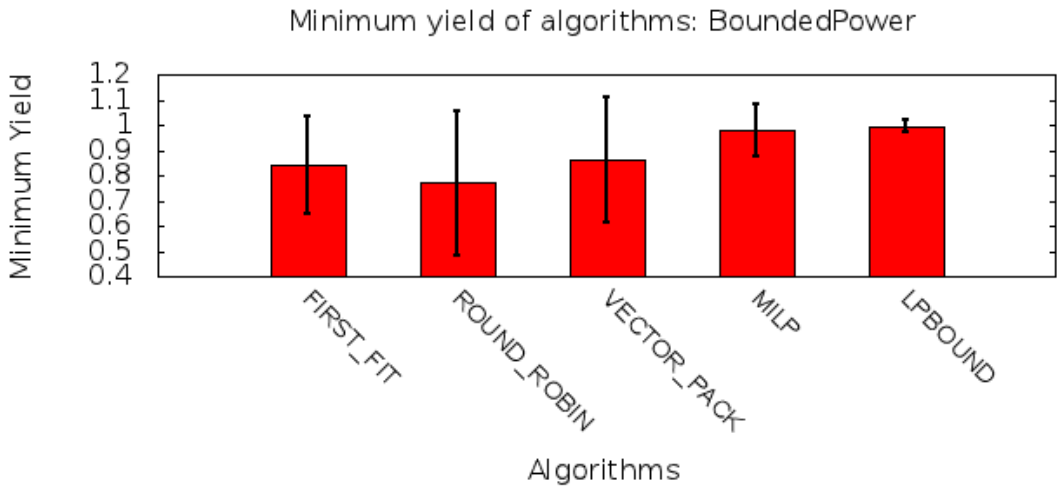


FIGURE 4.21 – Yield minimum résultant de l'allocation par les différents algorithmes

4.4.4 Ajout du refroidissement

Nous introduisons à présent l'infrastructure de refroidissement dans les expériences. Comme celle-ci est modélisée, chaque cluster aura un certain nombre de CRAC/CRAH, et ceux-ci seront liés à une partition des hôtes de ce cluster. Chaque équipement de refroidissement consommera évidemment de l'énergie, en fonction de si au moins un de ses hôtes est allumé. Nous génèrerons aléatoirement la consommation des CRAC/CRAH pour être entre 15% et 30% de la consommation totale des hôtes du cluster. Nous choisissons ces valeurs car comme montré dans le chapitre 1 en figure 1.1, l'infrastructure de refroidissement compte pour entre 15% et 30% de la consommation de l'infrastructure totale. La valeur en proportion de la taille du cluster, même si elle n'est pas totalement exacte, nous permet d'avoir des valeurs de consommation des CRAC/CRAH cohérentes. La prochaine expérience concerne donc l'introduction des infrastructures de refroidissement dans l'allocation de tâches. Cette fois-ci, nous enlevons l'hypothèse d'hétérogénéité pour y rajouter celle du refroidissement, afin de ne pas parasiter pour le moment les résultats. Comme précédemment, nous avons changé l'algorithme VP pour y introduire d'une part des considérations de refroidissement en allouant en groupe sur les hôtes reliés au même équipement de refroidissement. Nous appelons cet algorithme VP_COOL. L'autre variation de VP, que nous appellerons VP_CMAX_OVER_COOL tentera quand à elle d'allouer sur les hôtes qui ont le meilleur ratio C^{max} sur l'énergie consommée par l'équipement de refroidissement.

Comme nous pouvons le voir dans la figure 4.22, l'algorithme VP_COOL a les meilleurs résultats en terme d'énergie pour le problème BOUNDEDYIELD, alors que VP_CMAX_OVER_COOL a les mêmes résultats que VP, ce qui est dû au fait que les expériences sont ici faites sur un seul cluster, et que par conséquent tous les équipements de refroidissement consomment la même puissance. Pour le problème BOUNDEDPOWER, représenté en figure 4.23, nous avons des résultats similaires. Prendre en compte le refroidissement amène des améliorations en terme de performances et de consommation d'énergie.

Nous pouvons au final voir sur les figures 4.22 et 4.23 que nous pouvons gagner environ 2% de yield minimum en moyenne, ainsi que réduire la consommation d'énergie jusqu'à 11%. Cela est dû au fait que dans le cas du problème BOUNDEDYIELD nous allouons en prenant en compte

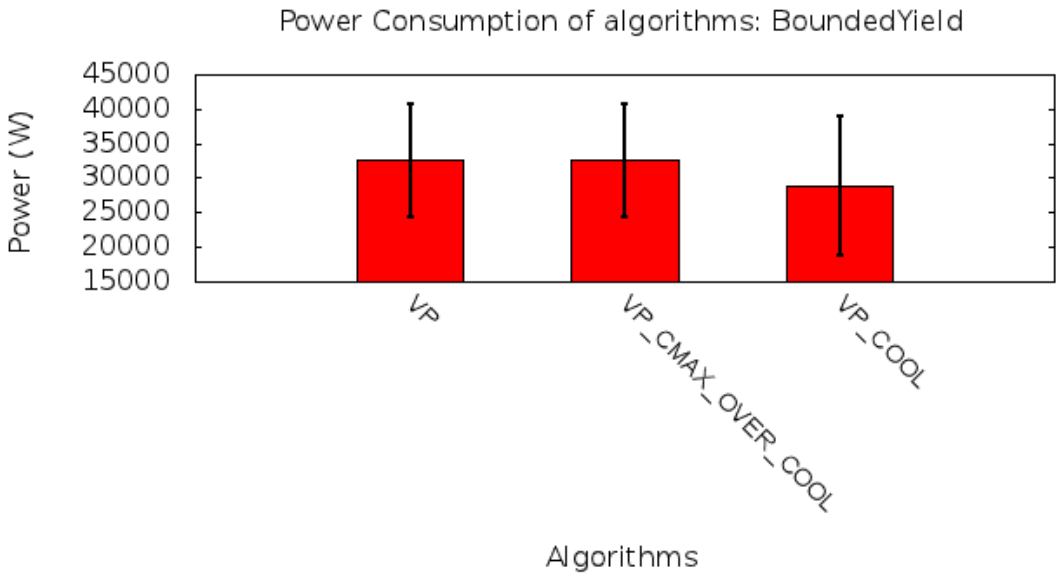


FIGURE 4.22 – Expérience sur le refroidissement : BOUNDEDYIELD

la consommation des CRAC/CRAH, et donc économisons de l'énergie. D'autre part, pour le problème BOUNDEDPOWER, comme allouer en fonction des infrastructures de refroidissement permet de réduire la consommation d'énergie, cela donne plus de marge de manoeuvre aux algorithmes, et permet par conséquent d'avoir de meilleurs résultats. Ainsi, prendre en compte le contexte dans lequel vont s'exécuter les différente tâches permet aux algorithmes de trouver des allocations efficaces.

Maintenant que nous avons pu voir l'impact de la prise en compte de l'hétérogénéité et de l'infrastructure de refroidissement de manière indépendante, il faut à présent vérifier de leur effet combiné. Pour ce faire, nous avons encore une fois décliné VP, afin de lui faire prendre en compte à la fois l'hétérogénéité et le refroidissement. Ainsi, nous aurons deux nouvelles déclinaisons : VP_H_COOL et VP_H_POWER. Le premier tente d'allouer en prenant en compte à la fois l'hétérogénéité et la puissance consommée par le refroidissement des machines, alors que le deuxième tente de prendre en compte l'hétérogénéité et la puissance totale inhérente à une machine.

La figure 4.24 nous montre la puissance consommée de tous les algorithmes, dans les cas où nous avons à la fois hétérogénéité et infrastructure de refroidissement. Comme nous pouvons le voir, les algorithmes qui nous donnent les meilleurs résultats sont ceux qui prennent les deux hypothèses en compte. Il y a une différence de presque 28% entre l'algorithme qui possède les pires résultats, et celui qui a les meilleurs.

Cependant, ces résultats ont un revers de médaille, puisque comme nous pouvons le voir sur la figure 4.25, ils viennent avec un coût. En effet, cette économie d'énergie vient avec une perte de performance de presque 12%.

On peut par contre remarquer sur la figure 4.25 que l'algorithme possédant le meilleur yield minimum est aussi celui qui donne les résultats les plus consistants, c'est-à-dire où l'écart type est le plus faible. La raison principale vient du fait qu'essayer d'allouer les différentes tâches sur les hôtes les plus efficaces limite notre capacité à garder les tâches elles-mêmes efficaces. C'est

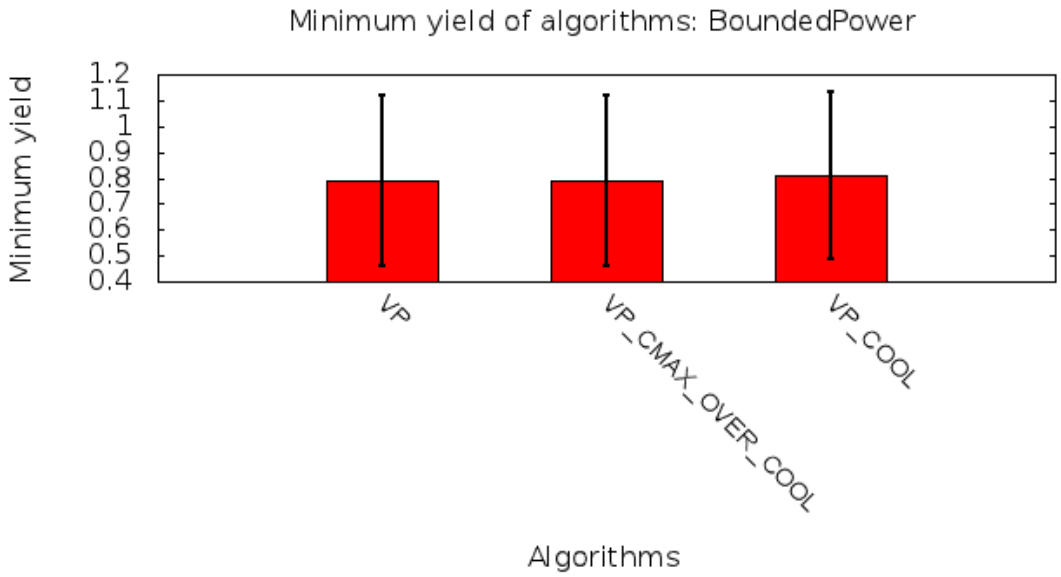


FIGURE 4.23 – Expérience sur le refroidissement : BOUNDEDPOWER

pourquoi il est encore une fois difficile de conclure sur quel algorithme nous donne globalement les meilleures performances en terme de compromis. Par contre, avoir des algorithmes autant préférenciels dans un domaine, nous permet d'être capable d'utiliser l'algorithme le plus adapté à la situation, et théoriquement être capable de réduire la consommation d'énergie de 28% pour le problème BOUNDEDYIELD, mais aussi être capable d'augmenter le yield de 16% pour l'autre problème.

Pour finir, nous regarderons les différents temps de calculs en fonction du nombre de tâches dans le système (à chaque fois avec un nombre d'hôte correspondant à environ 40% du nombre de tâches). C'est ce que montre la figure 4.26. Nous pouvons y voir que comme précédemment, les algorithmes gloutons allouent les tâches le plus rapidement (1000 tâches sur 400 hôtes en moins d'un quart de seconde). L'algorithme de vector packing quant à lui prend beaucoup plus de temps, mais reste toujours dans le domaine de l'acceptable.

4.5 Discussion, Conclusion, Ouverture

Les expériences précédentes ont bien montré une chose : il est important d'intégrer dans les algorithmes de gestion de tâches des considérations, même simplistes, de consommation d'énergie. Nous avons pu le voir, intégrer de telles considérations permet d'une part de réduire la consommation, mais aussi d'autre part les coûts. Cependant, puisque les différents buts sont antagonistes, souvent des économies en consommation d'énergies vont se répercuter sur la performance des tâches. Il faut donc être prudent vis-à-vis des choix que l'on fait, puisque ceux-ci vont directement conditionner la performance relative du système à l'arrivée.

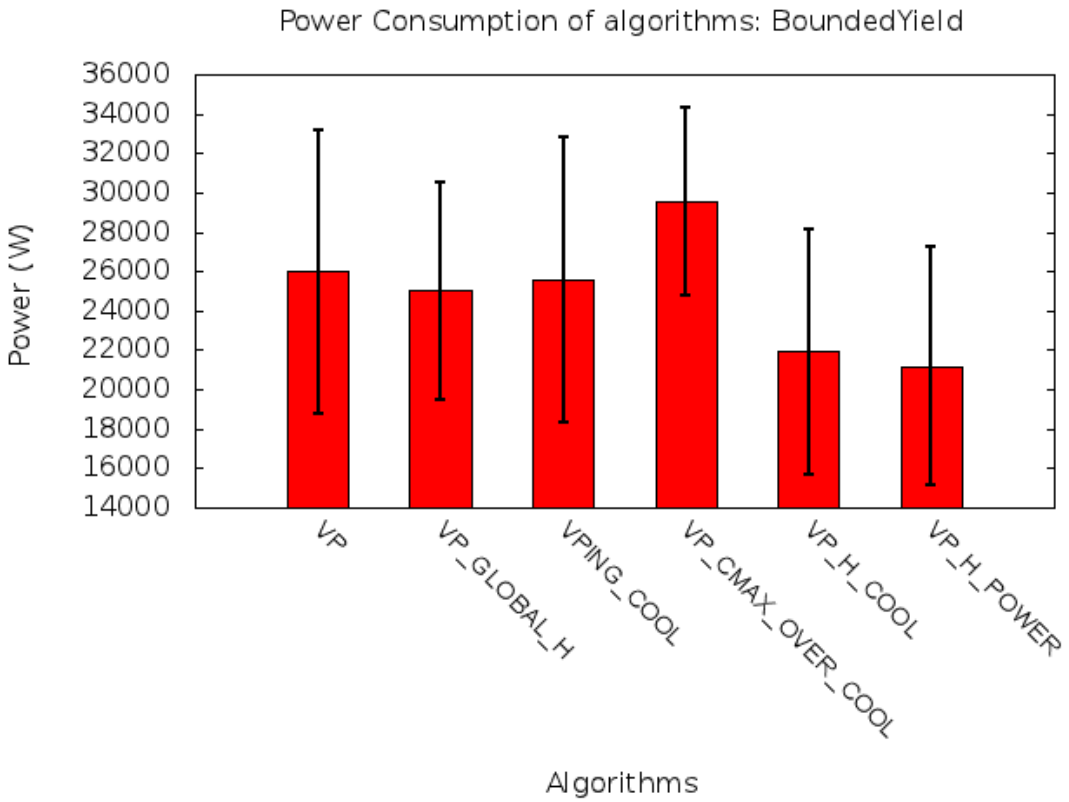


FIGURE 4.24 – Expérience sur l'hétérogénéité et le refroidissement combinés : BOUNDEDYIELD

4.5.1 Discussion sur les choix

Nous avons fait à ce propos de nombreux choix et hypothèses, induites ou non par les enseignements des expériences précédentes, certains ayant plus de conséquences que d'autres.

Non-utilisation du DVFS

Tout d'abord nous avons choisi de n'utiliser principalement que le levier de l'allumage et de l'extinction des hôtes pour économiser de l'énergie. Ce n'est pas le seul, et d'autres travaux ont montrés qu'il était possible de réduire la consommation du système par le biais par exemple du DVFS [105]. Seulement, même si nos expériences ne le montrent pas directement, ce levier peut souvent être en opposition à une consolidation efficace, puisque vis-à-vis de notre modèle, une allocation à une fréquence plus basse amène une capacité des hôtes plus petite, et par conséquent des tâches qui vont avoir moins de place, et donc utiliser plus de machines. Il est intéressant cependant de noter que le modèle est facilement extensible au DVFS, comme nous pouvons le voir dans [88].

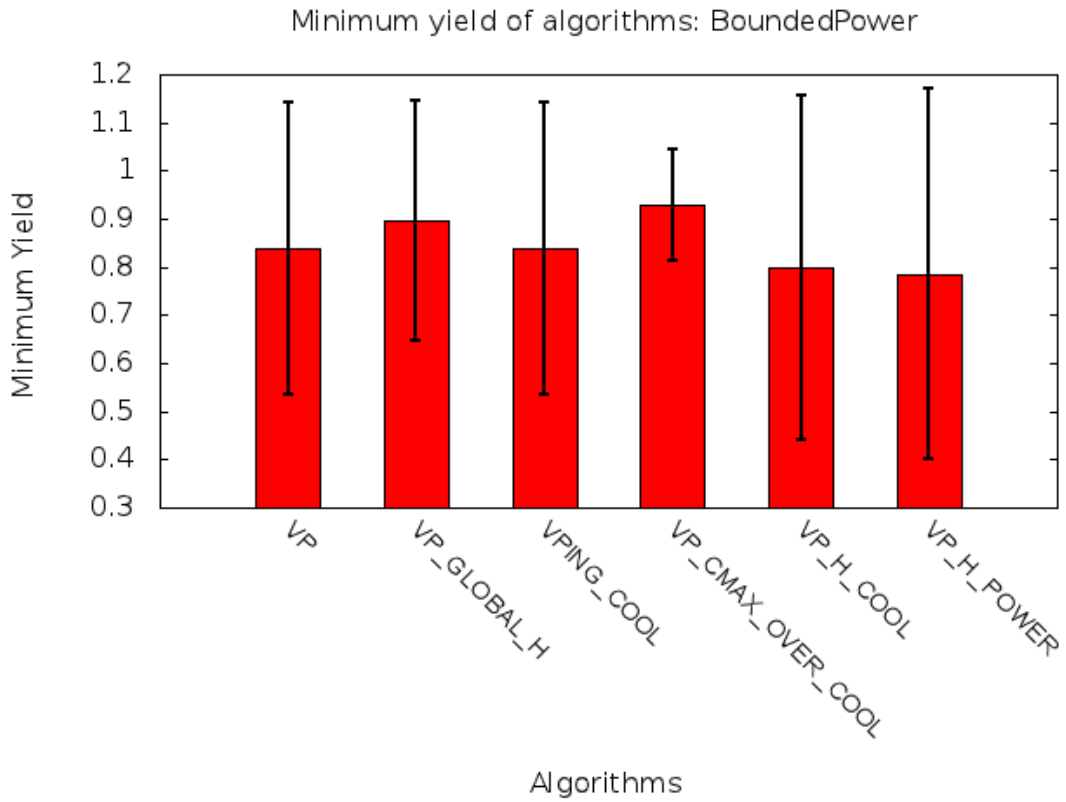


FIGURE 4.25 – Expérience sur l'hétérogénéité et le refroidissement combinés : BOUNDEDPOWER

Choix de la métrique de performance

Ensuite, les différentes fonctions objectif ont aussi fait l'objet de choix. Par exemple la métrique du yield minimum, qui est une métrique permettant de montrer à la fois la performance et l'équité, est celle que nous avons choisi. Pourtant, nous aurions tout aussi bien pu choisir d'optimiser le débit de tâches, en optimisant la somme des yields : $\sum_i \sum_h \frac{\alpha_{ih}}{\alpha_i}$. Cependant, cette métrique était plus adaptée à des systèmes qui tentent de faire le plus de tâches possibles sur une période donnée, et donc en contradiction avec certains objectifs, notamment garder une solution équitable. De la même manière, nous avions considéré évaluer l'impact des algorithmes sur le yield moyen.

Nous n'avons pas utilisé ces métriques pour des raisons évidentes d'équité. On imagine bien que faire une allocation augmentant au maximum le débit des tâches va tendre à favoriser les tâches courtes au détriment des longues. De même, faire une allocation optimisant le yield moyen, peut amener à creuser un écart important entre la tâche la moins bien allouée et la mieux allouée. On pourrait alors se demander si ces types d'allocations n'étaient pas à faire avec un SLA, une valeur minimale en dessous de laquelle on ne doit pas allouer les tâches, mais cette valeur au final se compare facilement au minimum yield, qui lui est une métrique capturant efficacement à la fois la notion de performance et d'équité.

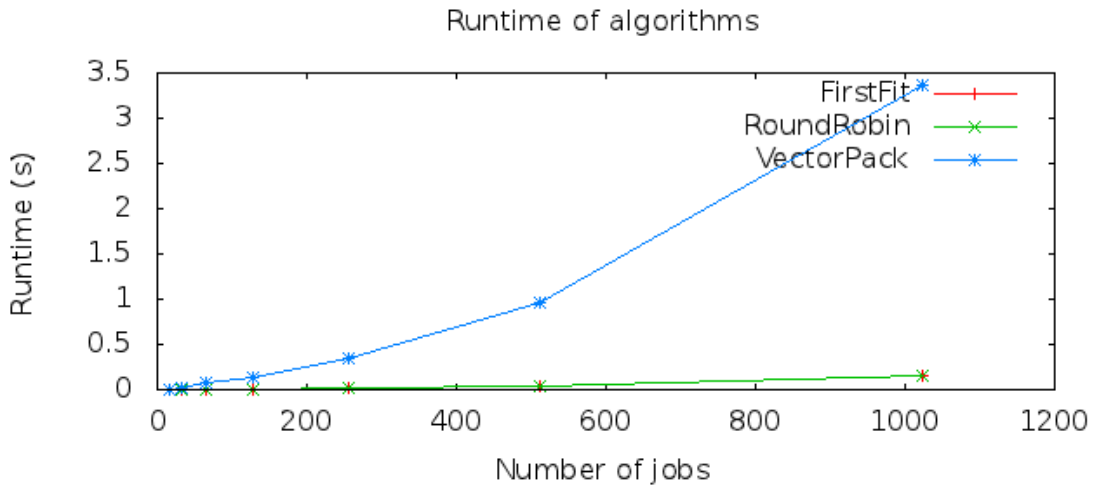


FIGURE 4.26 – Temps d'exécution des algorithmes

Choix du modèle de consommation d'énergie

Nous avons aussi fait des choix dans les autres fonctions objectif. Comme mentionné précédemment en section 4.2.1, nous avons choisi un système réducteur de la consommation d'énergie des hôtes, mais aussi des CRAC/CRAH. Cependant, seule la nature de la fonction, réduite à une expression linéaire, impacte les résultats. Changer pour un modèle plus précis dans le sens où nous ajoutons la nature polynomiale du calcul de la puissance et non juste en ajoutant une composante constante ou linéaire représentant une partie supplémentaire de la machine, nous enlève l'avantage de rester complètement dans une expression purement linéaire du problème, et nous enlève ainsi la capacité d'effectuer la résolution par solveur de programme linéaire et par conséquent de faire des comparaison avec l'allocation optimale, ou la borne rationnelle sur l'optimal. Il faut noter qu'il est possible d'affiner cet objectif en faisant par exemple du linéaire par morceaux afin d'être plus proche de la réalité. Nous avons cependant choisi de nous limiter au modèle linéaire par souci de simplicité d'une part, et d'autre part car il nous a paru plus important de nous concentrer sur d'autres problèmes comme le problème de la prise en compte globale de l'infrastructure, ou comme nous le verrons plus tard la prise en compte du coût induit par la migration lorsque nous passerons en dynamique.

Choix de la métrique de compromis

Le choix de la métrique de Z est peut être le plus épineux, dans la mesure où même si l'on peut anticiper le comportement de la métrique en fonction du choix de celle-ci, il sera quand même difficile de juger des résultats. C'est ce qui est par exemple arrivé lorsque nous avons défini cette métrique, au niveau du calcul de E^{max}, la valeur utilisée pour normaliser la partie exprimant l'énergie et qui exprime l'énergie maximale qui peut être consommée par un système. La formulation $E^{max} = \sum_h C^{max}$ n'est qu'une possibilité, nous aurions pu calculer une valeur telle que $E^{max} = \sum_h^{\lceil 2 \sum_i (\alpha_i) \rceil} (C_h^{max})$, qui représenterait une borne supérieure sur le cas où on allouerai 2 fois chaque tâches, après avoir trié les machines par C^{max} décroissant, et ainsi éviter certains cas d'allocations "naïves" dans lesquels la composante d'énergie disparait complètement. Cependant, les problèmes vis-à-vis de cette partie de la métrique ne viennent de E^{max} que pour

les allocations où par exemple nous avons 1 seule tâche dans un très grand nombre d'hôtes. Si l'on prend l'exemple où nous avons 1 seule tâche à allouer parmis 100 hôtes avec $C^{min} = 50$ et $C^{max} = 100$, avec l'expression de $E^{max} = \sum_h C_h^{max}$ nous aurons donc $E^{max} = 100 \times 100 = 10000$. Pourtant, l'allocation la pire possible consommera une puissance d'un seul hôte, soit 100W. La partie représentant l'énergie dans Z sera $(1 - \lambda)(E/E^{max}) = (1 - \lambda)(100/10000)$, ce qui rend difficile le compromis puisque la partie concernant l'énergie est trop petite par rapport à celle de la performance.

L'autre partie du problème vient surtout du fait de la normalisation des consommations d'énergie vis-à-vis de la performance qui est comprise entre 0 et 1. La nature affine de l'expression de la consommation d'un hôte pose problème, puisque d'une manière générale, diviser par E^{max} entrainera une valeur de la composante d'énergie d'un ordre moins important que la composante performance. Ceci est directement relié avec le choix de la valeur de λ, et son impact. Nous avons pu voir pour les expériences sur le problème MixedObjective que nous avions un impact fort de λ, mais cet impact n'était pas forcément ce à quoi on pouvait s'attendre. Nous avons pu voir que pour une grande partie de l'intervalle $[0, 1]$ de λ, il n'y avait que peu ou pas d'effets sur la composante d'énergie de Z. Les économies d'énergies significatives n'interviennent qu'à partir de $\lambda \in [0.7, 1]$. Ce qui remet en question l'application directe de la métrique décrite dans 4.3 et utilisée par EaResAlloc_Mixed, où nous avions utilisé une valeur de $\beta = \lambda$. C'est pour cette raison que les résultats aux valeurs de $\lambda = 0.5$ sont aussi peu concluantes. Il serait donc plus avisé de décrire un yield energy aware YE de la forme de Z, afin de pouvoir garder d'une part la formulation linéaire de Z, mais aussi de pouvoir faire correspondre directement β et λ pour s'approcher plus facilement des effets recherchés sur l'allocation de tâches.

On peut donc se questionner sur l'utilité d'avoir un éventail de valeur pour λ entre 0 et 1. Cependant, puisque le choix de la valeur de λ est problématique, il sera opportun, quel qu'en soit l'intervalle, de laisser l'ajustement du λ à un algorithme, qui pourra par exemple ajuster cette valeur en fonction des allocations effectuées. Il ne sert donc à rien de se limiter à certaines valeurs de λ et il sera plus intéressant de pouvoir choisir un éventail de valeur plus large.

4.5.2 Discussion sur les compromis

Cette partie d'expérimentation montre bien aussi, on ne le répètera jamais assez, l'utilité et les dangers de faire des compromis d'un côté, pour gagner de l'autre. Nous voyons bien qu'il est intéressant d'économiser de l'énergie, en sacrifiant un peu de performances, alors qu'il sera souvent risqué de pousser un peu plus la réduction de la consommation d'énergie, car cela pourra amener des pertes bien trop importantes en performances, si cela n'est pas fait comme il faut. C'est ce que nous avons pu voir dans l'expérience montrée en figures 4.18 et 4.17, où l'on voit bien l'impact que peuvent avoir des tendances différentes des algorithmes à l'optimisation, puisque certains algorithmes ont trop tendance à favoriser la réduction de la consommation d'énergie au détriment de la performance, alors que d'autres algorithmes auront plutôt tendance à trop favoriser la performance, au risque de ne pas trouver d'allocation valide.

Cependant, le compromis énergie/performance n'est pas le seul à entrer en compte. Lorsque l'on ajoute des constraintes transversales comme l'hétérogénéité, nous avons pu observer des comportements nécessitant de prendre en compte ces contraintes, afin de pouvoir capitaliser sur ceux-ci pour faire des économies d'énergie à moindre coût. Ce faisant, nous avons pu nous rendre compte que l'hétérogénéité des hôtes n'avait pas un effet allant dans le sens de l'effet des infrastructures de refroidissement, c'est-à-dire que dans certains cas, favoriser une allocation de tâche sur l'hôte ayant la plus grande capacité de calcul et la meilleure efficacité énergétique peut être moins bonne que d'allouer cette tâche sur un hôte moins efficace qui permettra d'éteindre un équipement de refroidissement.

Puisque l'hétérogénéité n'a pas le même effet sur l'allocation des tâches que l'infrastructure de refroidissement, il apparait que dans certains cas, ces deux effets seront antagonistes. Ainsi, il sera plus efficace de se concentrer sur une des deux contraintes, souvent celle qui amènera le plus d'économies d'énergie. Cependant, certaines typologies d'instances, comme par exemple celles qui sont surchargées, nécessiterons un traitement particulier, car il faudra pour celles-ci ne pas se concentrer sur l'extinction d'un CRAC/CRAH, puisqu'il sera souvent impossible de respecter la borne sur la performance tout en éteignant un CRAC entier.

Cela montre encore une fois la difficulté de trouver un compromis, lorsque l'on ajoute des contraintes supplémentaires, celles-ci sont interdépendantes, et amènent à des situations différentes. Cependant, comme nous l'avons montré, avec des algorithmes simples, il sera intéressant d'utiliser une batterie d'algorithmes simples, à utiliser dans différents cas, pour arriver à des allocations performantes.

4.5.3 Conclusion

Dans ce chapitre, nous avons défini le problème de l'allocation de services efficace en énergie dans un cadre statique. Pour cela, nous avons utilisé une modélisation par un ensemble de contraintes linéaires, ce qui nous a permis d'avoir un modèle facilement extensible d'une part à l'hétérogénéité qu'il existe à l'intérieur d'un cluster, et entre les clusters, et d'autre part la gestion des infrastructures de refroidissement. À partir de ce modèle, nous avons séparé le problème de l'allocation en trois problèmes distincts, le problème borné en performance, où l'on tente de minimiser la consommation d'énergie en respectant une performance minimum sur les service. Ensuite le problème borné en consommation d'énergie, où ici le but était de maximiser la performance, sous contrainte de consommation de puissance. Enfin, le problème du compromis, qui intervient lorsque l'on n'a ni contrainte en puissance, ni en performance, et où l'on tente de maximiser la performance en réduisant la consommation d'énergie.

Nous avons ensuite défini différents algorithmes pour répondre aux différents problèmes et prendre en compte certains aspects du problème. Nous avons montré que prendre en compte l'énergie permettait des économies d'énergie significatives sous contrainte de performance. Nous avons par la suite étudié l'effet que pouvait avoir l'hétérogénéité sur ces différentes métriques, ainsi que les CRAC/CRAH sur la consommation d'énergie. Il en ressort que prendre ces aspects en compte nous permet de mieux appréhender le système étudié, et d'effectuer des allocations plus précises. Certains problèmes discutés précédemment sont cependant soulevés, comme l'antagonisme qu'il peut exister entre la prise en compte du refroidissement et la prise en compte de l'hétérogénéité. Il apparait notamment que selon le système étudié, et surtout la part de consommation de puissance que prend un CRAC/CRAH par rapport à la consommation des hôtes, il faudra favoriser soit l'allocation sur les machines efficaces, soit l'allocation permettant d'éteindre un équipement de refroidissement.

Nous avons pu tirer un certain nombre d'enseignements des différentes expériences que nous avons conduit, concernant les différents choix qu'il est possible de faire lors de l'allocation de tâches. Premièrement, nous avons vu que l'ordre d'allocation des machines virtuelles ainsi que l'ordre des hôtes sur lesquels nous allouons était important. En effet, les expériences montrent qu'il est plus intéressant d'allouer les tâches avec les plus gros besoins en premier, afin d'augmenter le taux de succès des algorithmes. De plus, en fonction des caractéristiques des hôtes, il est plus intéressant d'allouer sur les machines ayant le plus petit C_h^{max} en premier, tout simplement car cela permettra de choisir a priori les hôtes les plus efficaces énergétiquement parlant en premier. Cependant, dans les expériences que nous avons faites précédemment, nous avons pu constater qu'allouer en fonction du surtoût induit par l'exécution des tâches peut être intéressant aussi. Cela dit, il dépendra surtout du rapport entre C_h^{max} et $C_h^{max} - C_h^{min}$, ce qui définira

l'importance relative des paramètres, et permettra un choix entre les deux.

De plus, concernant l'hétérogénéité et le refroidissement, s'il est important de prendre les deux en compte, il faudra prendre aussi en compte la propension du système à être capable d'éteindre des équipements de refroidissement. Si le cluster ne possède que peu de ceux-ci par exemple, il ne sera pas judicieux de tenter de les éteindre, puisque cela n'arrivera que peu souvent. Par contre, si le cluster est large, il faudra prendre en compte le refroidissement en priorité afin d'économiser de l'énergie. Cela étant dit, il sera donc bon de prendre en compte la finalité des actions sur le refroidissement, à savoir si l'on réussit à éteindre ou non un CRAH, afin de pouvoir par exemple refaire une allocation sur seulement les hôtes liés au CRAH que l'on laissera allumés.

En pratique, au niveau du data center lui même, il pourra être intéressant de prendre en compte ces différents effets pour construire son data center. Avoir un large data center composé de plusieurs clusters modulaires autosuffisants sera une bonne étape dans ce sens par exemple. Cela nécessitera aussi de définir et de fournir les infrastructures nécessaires aux algorithmes de placement pour pouvoir avoir des résultats encore meilleurs.

Il ressort enfin des expériences par simulations que nous avons effectué, qu'une bonne prise en compte des différents aspects dans les deux problèmes mono objectif (BoundedYield et BoundedPower) permet à nos algorithmes d'avoir des bons résultats, comparé à des algorithmes gloutons tels que FirstFit. Pour le problème MixedObjective par contre, le problème est comme mentionné précédemment plus épineux, et les résultats sont souvent trop mitigés pour pouvoir en tirer des conclusions. Cependant, ces travaux ouvrent différentes perspectives de recherche dans différentes directions.

4.5.4 Ouverture

Il est encore possible, et nécessaire, d'augmenter le modèle pour être au plus près du comportement des infrastructures réelles. Beaucoup de chemin reste à faire, tant au niveau des hôtes, de l'infrastructure, qu'au niveau des tâches. En effet, même si nous laissons de côté le fait que nous présupposons avoir connaissance à priori des demandes de tâches, sous la forme par exemple de SLA, il reste évident que ces tâches peuvent avoir des contraintes supplémentaires.

Par exemple, nous avons montré qu'il était plus intéressant de faire une allocation globale, plutôt que de faire une allocation par cluster. Cela posera un problème si nous prenons en compte l'allocation de tâches interdépendantes. Dans le cas où deux tâches ont besoin l'une de l'autre pour fonctionner, nous serons souvent devant le problème d'une forte réduction de la performance si celles-ci ne sont pas sur le même cluster. Il faudra donc allouer les tâches en tant que groupes pour palier à ce problème, ce qui peut être géré par l'algorithme d'allocation.

L'algorithme d'allocation en tant que tel est aussi améliorable dans le sens où, comme nous avons pu le voir, la trop forte consolidation de EaResAlloc_BoundPower peut porter préjudice dans certains cas, puisqu'elle pourra amener à ne pas être capable de respecter la borne, et ainsi amener à des performances laissant à désirer, autant au niveau des performances lorsque l'on va trop consolider au détriment des performances, qu'au niveau de la fiabilité de l'algorithme, lorsque l'on échouera trop souvent.

D'une manière générale, il sera intéressant d'augmenter d'une part les contraintes, et d'autre par la gestion de ces contraintes par les algorithmes. Dans tous les cas, l'allocation de tâches statique est considérée comme initiale, et peut être considérée comme périodique, mais cette allocation sera à faire dans un environnement virtualisé, reposant notamment sur la migration possible des tâches. Cette migration est problématique, puisque même en posant des améliorations à l'infrastructure comme une gestion centralisée des images des machines virtuelles, le temps de la migration sera d'importance maximale. En effet pour être capable d'allouer et surtout de réallouer les tâches avec une période suffisamment courte pour pouvoir réagir aux différentes

variations du système, il faudra bien prendre en compte le surcoût de la migration. C'est à cet endroit que nous voyons les limites de l'approche statique, et que nous devons passer à la réallocation dynamique pour pouvoir mieux gérer un système.

Il serait aussi bon de rajouter des types de ressources que vont consommer les tâches dans les expérimentations que nous avons fait. Nous avons fait le choix de n'utiliser seulement une ressource fluide, le CPU, et une ressource rigide, la RAM. Pourtant, comme mentionné précédemment, le modèle est généralisable à un ensemble de ressource rigides et ressources fluides. Il conviendrait ainsi d'expérimenter en rajoutant deux ressources communément utilisées dans la définition des tâches, la consommation en disque dur en tant que ressource rigide, et la consommation en bande passante en tant que ressource fluide. Nous pourrions ainsi voir si les résultats sont les mêmes d'une part, mais aussi si des adaptations du modèle doivent être faites afin d'améliorer la performance globale des allocations.

Chapitre 5

De l'Allocation Statique Vers la Réallocation

Nous avons vu dans le chapitre précédent comment allouer de manière efficace des tâches sur des machines afin de réduire la consommation d'énergie et d'optimiser la performance. Nous avons aussi soulevé le problème que cette allocation est statique, et par conséquent adaptée aux charges peu variables au cours du temps. Seulement dans le monde du cloud computing, il est souvent intéressant d'être capable de réagir à la demande à un instant donné. Pour ce faire, il faudra utiliser la migration (live, c'est-à-dire sans interruption de service) fournie par l'infrastructure de virtualisation.

Seulement, comme nous allons le voir, cette migration n'est pas sans poser des problèmes qu'il va falloir résoudre pour pouvoir utiliser ce type d'approche.

5.1 Passer du statique au dynamique

La dynamicité en général pose la question de la gestion du temps dans le modèle. Si l'on prend un exemple simple d'une machine qui doit s'allumer et s'éteindre, lorsque l'on est en statique avec des temps suffisamment grands, on peut se permettre de négliger le temps d'allumage. Cela devient de moins en moins le cas lorsque l'on réduit l'intervalle de temps entre deux allocations.

5.1.1 Pourquoi ?

Afin de mieux comprendre les raisons derrière l'importance non seulement de prendre en compte la dynamicité du système, mais aussi de la modéliser de façon cohérente, reprenons l'exemple d'une machine qui doit s'allumer et s'éteindre. Lorsque nous étions en allocation statique, le temps entre les différentes allocations était suffisamment grand pour pouvoir négliger le temps d'allumage et d'extinction des hôtes. Si l'on doit allouer et réallouer dynamiquement les tâches, il faut prendre en compte ce temps d'allumage et d'extinction, car pendant ceux-ci, les machines seront inutilisables, mais consommeront des ressources. Cette consommation de ressource est inévitable lorsque l'on éteint des machines que l'on va devoir plus tard rallumer quand la charge aura changé.

En plus d'être inutilisables pendant un certain laps de temps, il faudra changer la façon dont on approche la gestion des machines, puisque éteindre et rallumer trop souvent un hôte induira un surcoût de consommation que nous pourrions éviter. Il ne faut donc plus se limiter à éteindre

les hôtes lorsque ceux-ci sont inutilisés, mais en garder certains allumés même lorsqu'ils sont peu chargés afin de pouvoir amortir un changement dans les demandes des tâches.

Le problème n'est pas seulement au niveau de l'allumage et de l'extinction des hôtes, mais il est aussi, et surtout, au niveau de la migration induite par la réallocation. On pourrait penser qu'il suffit d'effectuer des allocations successives lorsque les demandes des tâches du système dévient suffisamment des demandes utilisées lors de l'allocation précédente. Seulement la migration d'une machine virtuelle depuis une machine physique vers une autre prend non seulement du temps, mais consomme aussi des ressources, à la fois sur la machine source que sur la machine destination, mais aussi sur l'infrastructure réseau. Rappelons-le, la migration des machines virtuelles telle que définie dans Xen[18] ou KVM[65] par exemple fonctionne en transferant la mémoire depuis la machine physique source vers la machine physique destination. Pendant le transfert, la machine virtuelle source fonctionne toujours, ce qui amène à avoir une partie de la mémoire qui a changé pendant le transfert. Un nouveau transfert de ce qui a été changé est fait, et l'on répète ces deux opérations un certain nombre de fois ou jusqu'à avoir un noyau suffisamment réduit de mémoire restante. À ce moment, la machine virtuelle source est stoppée, la mémoire restante transférée, et la machine virtuelle destination démarre.

On peut déduire de ce fonctionnement que la migration de machine virtuelle met une pression importante sur l'infrastructure réseau. Ainsi, si l'on prenait la solution consistant à faire des allocations successives complètes, on aurait souvent un grand nombre de migrations, ce qui induit un goulot d'étranglement au niveau de l'infrastructure réseau, et qui va ralentir fortement la complétion de l'allocation, à tel point qu'il peut arriver des situations où l'allocation se termine dans un système où les besoins des tâches ont changé entre le début et la fin de la migration, nécessitant ainsi une nouvelle allocation.

Pour prendre en compte la dynamicité du système, nous devons donc prendre en compte le fait que la migration consomme des ressources, et surtout plutôt que des allocations successives de toutes les tâches, il faudra faire des allocations successives en prenant en compte l'état courant du système. Plutôt que d'allouer des tâches, il faudra les réallouer, c'est-à-dire migrer une machine virtuelle d'une source vers une destination.

Si nous prenons enfin le point de vue énergétique de la migration, la migration d'une machine virtuelle va consommer de l'énergie sur le réseau, ce qui peut-être considéré comme un surcoût constant fonction de la taille de la machine virtuelle, mais aussi des ressources sur à la fois la machine source et la machine destination. En effet, le transfert de données consomme à la fois des ressources CPU et réseau sur les machines physiques, mais aussi des ressources CPU pour initialiser la machine virtuelle destination et pour arrêter la machine virtuelle source.

Au vu de ces différents problèmes apportés par la migration et par la dynamicité en général, il faut étendre le modèle présenté dans les chapitres précédents, pour y intégrer les effets décrits ci-dessus.

À l'instar du modèle statique, le passage à la dynamicité va demander une gestion un peu différente de la qualité de service. Si dans le chapitre 4, nous étions concentrés sur l'augmentation de la performance globale des tâches pour l'allocation initiale, nous allons devoir ici, du fait de la variabilité des tâches et de la dynamicité du système, prendre en compte non plus une maximisation de performance pour les tâches, mais une minimisation du gaspillage de ressources, c'est-à-dire que nous voudrons effectuer des allocations au plus près de ce que la tâche consomme réellement. Il faudra donc effectuer des allocations permettant de toujours satisfaire les tâches, quelque soit les ressources qu'elles consomment. Nous utiliserons donc des *Service Level Agreements* (SLA) pour déterminer si une tâche est satisfaite ou non, et nous allouerons les tâches en fonction de ces SLA.

Le parallèle entre SLA et borne sur le yield a été fait dans la partie précédente, et il convient de le répéter ici. Si nous considérons que les besoins en ressource d'une tâche forment ce qui a été

accepté pour accord entre le client et le fournisseur de service, une allocation en dessous de cette borne constitue une violation de SLA. Dans la partie précédente, nous considérions qu'une telle violation impliquait forcément un rejet de l'allocation dans le cas du problème BOUNDEDYIELD, c'est-à-dire que si nous ne pouvions trouver une allocation nous permettant d'avoir au minimum $x\%$ de satisfaction pour chacune des tâches, l'algorithme utilisé pour l'allocation n'avait pas trouvé une solution satisfaisante. Ici, puisque nous avons plusieurs allocations et réallocations successives, nous allons pouvoir changer cette valeur x, même si celle-ci n'a pas trouvé de solution. Ce qui veut dire que si nous ne trouvons pas de solution satisfaisante, nous considérerons l'allocation résultante comme non satisfaisante et appliquerons les pénalités correspondantes, et nous passerons tout de même à l'allocation suivante. Une description de la définition des SLA est donné plus tard en partie 5.3.1, ainsi qu'une explication plus détaillée des différences du modèle de SLA en statique et dynamique en partie 5.1.2.

5.1.2 Extension du modèle

La première chose pour introduire de la dynamique dans le système est d'introduire une notion de temps. Nous possédons une infrastructure de cloud qui doit être gérée par une entité centralisée appelée gestionnaire autonome (ou *autonomic manager*). Ce gestionnaire autonome va réagir aux différents états du système en effectuant un certain nombre d'actions. Pour ceci, nous définissons le gestionnaire autonome comme suivant la très répandue boucle autonome de contrôle appelée MAPE-K, montrée en figure 5.1. Le **M** veut dire *Monitoring*, pour la surveillance des différentes métriques pertinentes de notre infrastructure. Le **A** veut dire *Analyze* puisqu'on va analyser les données du monitoring, alors que le **P** veut dire *Plan*, pour planifier les actions proactives à effectuer sur le système. Le **E** est pour *Execute*, qui représente l'exécution des actions définies. Enfin, le tout s'axe autour du **K**, pour *Knoweledge*, qui représente la base de connaissance alimentée par le monitoring et l'exécution, sur laquelle s'appuie l'analyse.

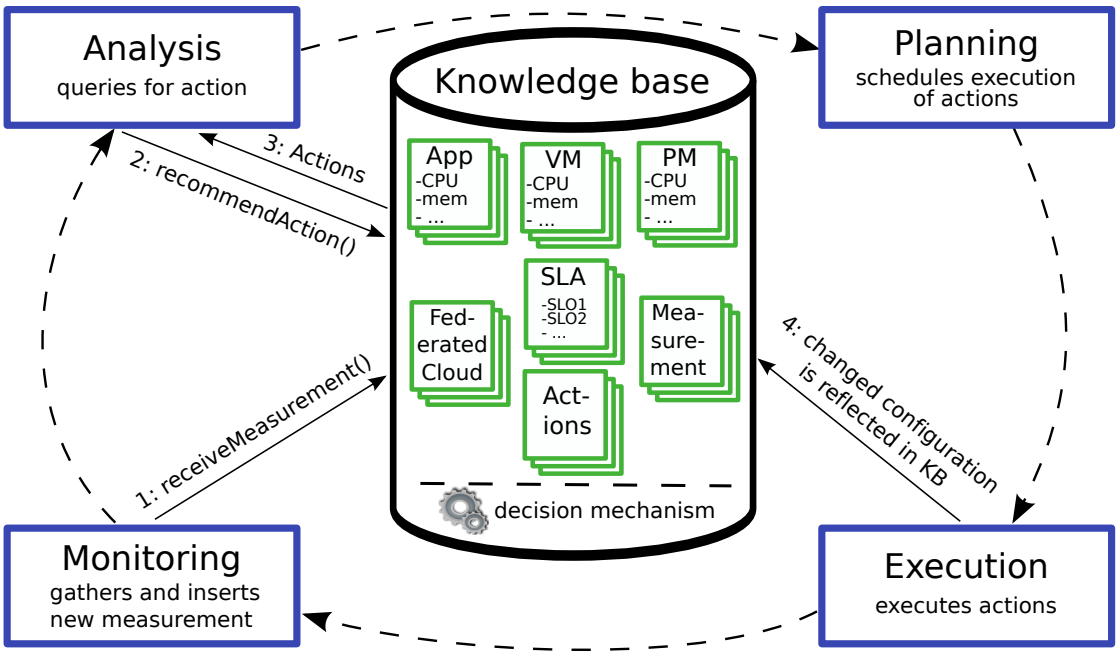


FIGURE 5.1 – Boucle autonomique MAPE-K

Ce fonctionnement en boucle autonome nous permet non pas de définir du temps à proprement parler, mais des intervalles de temps, certes arbitraires, pendant lesquels on effectuera une révolution complète de cette boucle. Le temps ne sera donc pas défini en secondes, ou minutes, mais en intervalle de temps, ou *timesteps*, pour nous permettre de définir un comportement sur un intervalle de temps, sans avoir de définition précise du comportement réelle de l'infrastructure. En effet, il est difficile de prédire à l'avance le temps de migration d'une machine virtuelle, comme il est difficile de prédire le temps de démarrage ou d'extinction d'un serveur. Cependant, définir des intervalles de temps nous permet d'avoir une valeur que l'on peut régler, mais aussi de partir sur une borne supérieure dans les temps, si nous définissons la migration comme prenant le même temps que l'extinction d'un hôte.

Nous devons donc définir trois choses : le comportement de la migration, de l'allumage d'un hôte, et de son extinction.

Commençons par l'allumage et l'extinction d'un hôte. Ces actions prennent du temps d'une part comme nous l'avons vu précédemment, mais consomment aussi des ressources machines d'autre part. Nous définissons donc l'allumage et l'extinction comme prenant respectivement *startup_time* et *shutdown_time* (les deux valeurs définies à 1 pour le moment), et consommant la totalité des ressources de la machine pendant ce temps. C'est ce que montre les figures 5.2 et 5.3. La figure 5.2 nous montre le modèle pour l'allumage d'un hôte. Lorsque l'on prend la décision d'allumer une machine, entre le timestep t et t+1, la machine ainsi choisie sera allumée et consommera de l'énergie comme si elle était chargée à 100%, mais ne sera pas utilisable pour allouer des tâches. Elle sera finalement utilisable pour l'allocation seulement à partir de t+2 (si la durée de migration est définie à 1 timestep). De la même manière, la figure 5.3 nous montre le modèle pour l'extinction d'une machine. Comme pour l'allumage, une fois la décision prise, la machine sera inutilisable et chargée à 100%, et sera seulement utilisable au timestep t+2.

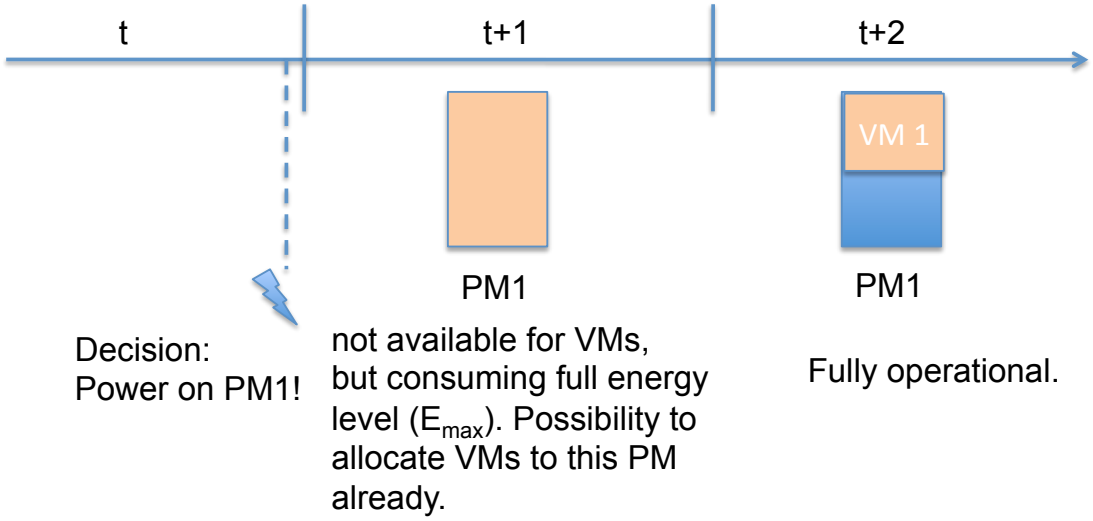


FIGURE 5.2 – Modèle d'allumage d'une machine physique

Au vu de notre modèle de consommation de puissance, cela veut dire qu'un hôte qui s'allume ou qui s'éteint consommera 100% de la puissance CPU. Pendant ce temps, la machine est inutilisable, c'est-à-dire que l'on ne peut pas y affecter de tâches. Cela veut aussi dire qu'il y a de l'inertie dans l'allumage ou l'extinction des hôtes, dans le sens où si l'on décide d'éteindre une machine à tort, et que l'on veut la rallumer, il faudra un timestep pour qu'elle s'éteigne, et un

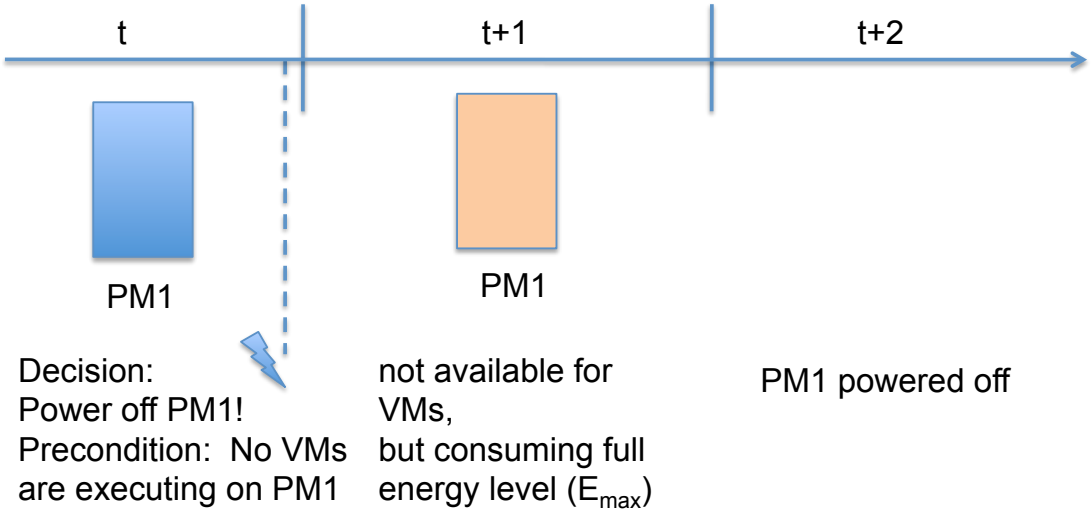


FIGURE 5.3 – Modèle d'extinction d'une machine physique

timestep pour qu'elle se rallume, et nous ne pourrons donc l'utiliser qu'à partir de 2 timesteps plus tard. De plus, pendant ces deux timesteps, la machine aura été allumée au maximum de sa charge, et donc aura consommé le maximum d'énergie possible.

Nous devons à présent choisir le modèle de migration, ou plus précisément le modèle de coût de la migration dans notre cloud. Comme mentionné précédemment, la migration est problématique car elle consomme des ressources à la fois sur l'hôte source et destination, mais aussi prend du temps. Or, du fait de son fonctionnement, surtout pour les migrations de type pré copie, le temps de migration ainsi que le volume transferré et donc une partie des ressources consommées seront déterminé par un ensemble de caractéristiques inhérentes à la machine virtuelle et à l'application s'exécutant à l'intérieur.

Un certain nombre de travaux ont été effectués pour tenter de modéliser la migration de machines virtuelles. C'est le cas par exemple de Breitgand et al. dans [26], où les auteurs définissent un modèle de migration pour les migrations qui utilisent une partie de la bande passante allouée aux machines virtuelles. Ce modèle dépend donc d'une variable qui est la partie de bande passante appartenant au service, et implique par conséquent une interdépendance entre les variables à optimiser pour la performance, comme la bande passante, et la performance perdue pendant la migration.

D'autres travaux ont modélisé la performance de la migration de machines virtuelles comme dépendant de la taille de la machine virtuelle, le taux d'invalidation des pages mémoires, et la bande passante du lien réseau de la machine physique [76, 9]. Les auteurs dans [76] définissent ensuite la consommation énergétique issue de la migration d'une machine virtuelle. Cependant, bien que ces modèles soient relativement précis pour définir le coût d'une migration, il apparaît qu'il soit difficile de connaitre ces paramètres. En effet, le taux d'invalidation des pages dépendra fortement du type d'application s'exécutant à l'intérieur de la machine virtuelle, et un système comme celui auquel nous nous intéressons, qui n'a pas de connaissance à priori des applications, ne pourra pas prédire avec certitude ce paramètre, et donc l'effet de la migration. De plus, ces modèles sont fiables pour peu de migrations concurrentes, car un grand nombrede migrations concurrentes aura pour effet de déclencher la fin de la migration, l'étape *stop-and-copy*, et bornera donc la migration.

D'autres travaux tels que [30], utilisent une approche stochastique pour modéliser la migration, dans le cadre notamment d'applications temps réelle douce.

Dans la majorité des travaux cependant, notamment ceux portant sur les économies d'énergie dans les clouds, une approche simplifiée est utilisée, en prenant en compte la migration comme un coût. C'est le cas par exemple dans [31], où les auteurs utilisent le coût en transfer réseau de la machine virtuelle, modélisé en tant que taille de celle-ci, pour donner un poids à la migration. Dans [38] par contre, les auteurs utilisent plutôt le nombre de migrations pour tenter d'en limiter le nombre. Dans [87] enfin, les auteurs donnent un coût énergétique aux migrations, pour ensuite tenter de limiter l'énergie sous contrainte de performance.

Nous pouvons ainsi le voir, la migration peut être prise en compte de maintes manières, et il apparait cohérent de la prendre en compte. Par contre, il est difficile de modéliser avec précision la migration dans un système où l'on ne connait pas avec exactitude les applications. C'est pourquoi nous avons choisi un modèle de migration pessimiste basé sur un temps arbitraire certe, mais souvent supérieur à la réalité.

Prenons maintenant le modèle choisi pour définir la migration. Celui-ci doit se conformer à deux contraintes : les tâches qui migrent doivent consommer des ressources à la fois sur la machine physique source et destination, et la migration doit prendre du temps. Ainsi, nous définirons une tâche qui migre comme montré par la figure 5.4. Lorsqu'une tâche migre, elle consommera le même montant de ressource sur l'hôte source et destination, le montant de ressource alloué par l'algorithme d'allocation/réallocation, et ce pendant *migration_time* timesteps (défini à 1 pour commencer). De la même manière que pour l'allumage et l'extinction des hôtes, il y aura une certaine inertie et surconsommation liée à la migration, il faudra donc faire attention à ne pas faire de migration non nécessaire.

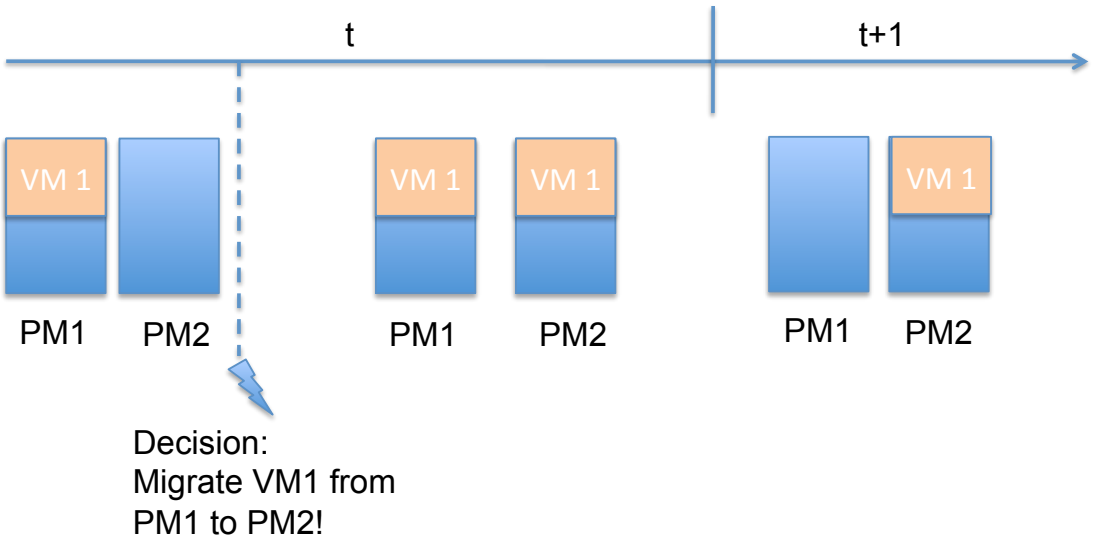


FIGURE 5.4 – Modèle de la migration d'une machine virtuelle

Pour résumer les actions possibles, prenons comme représenté dans la figure 5.5 un cloud composé de 2 machines physiques (H1, H2) et 2 machines virtuelles(VM1, VM2). À l'intervalle t les deux machines virtuelles sont allouées sur H1, alors que H2 est éteint. Àu passage de l'intervalle t à $t+1$ les besoins ont augmenté et nécessitent l'allumage d'un deuxième hôte, H2 qui est en allumage. Àu début de l'intervalle $t+2$, H2 a fini de s'allumer et est utilisable, on peut

donc commencer la migration de VM1 depuis H1 vers H2. On remarquera la consommation de ressources identique sur les deux hôtes. À l'intervalle $t+3$ la migration a fini et chaque VM est sur un hôte différent. À l'intervalle $t+4$ les besoins de la tâche VM1 ont diminué et nous voulons économiser de l'énergie, nous faisons donc migrer la tâche VM1 depuis H2 vers H1. À l'intervalle $t+5$ enfin, la migration est finie et nous commençons à éteindre l'hôte H2, qui aura fini de s'éteindre à la fin de $t+5$. On notera ici que la prise des décisions d'allumage/extinction/migration se fait à la limite entre deux timesteps. Cela veut dire que la décision se fera sur les mesures prises à la fin du timestep qui vient de finir, et sur la conaissance de l'état du système au timestep suivant, c'est-à-dire quelles migrations sont finies, et quels hôtes sont disponibles.

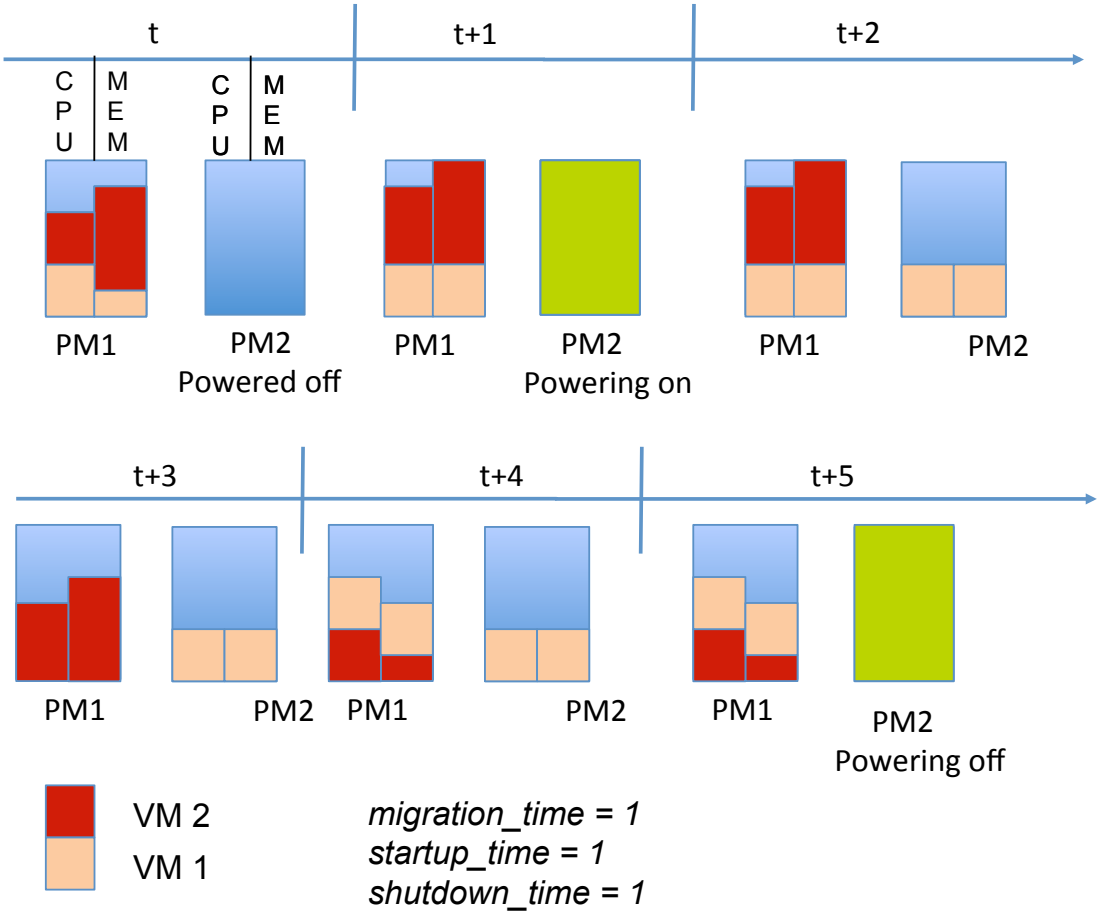


FIGURE 5.5 – Configuration possible d'un cloud sur 6 timesteps

5.2 Nouvel ensemble de contraintes

Maintenant que nous avons défini les différents aspects nécessaires à définir la réallocation dynamique dans le cloud, il faut à présent intégrer ces différents concepts dans le modèle d'ensemble de contraintes. Pour ce faire, nous définissons un ensemble de quatre variables binaires représentant l'allumage, l'extinction des hôtes, et la migration depuis et vers un hôte. Ces variables sont

définies binaires et non rationnelles car nous ne pouvons pas avoir un hôte à moitié en train de s'allumer ou de s'éteindre, de la même manière qu'une tâche ne peut être en partie en train de migrer. Rappelons que nous ne considérerons pas ici ni l'hétérogénéité des machines, ni les infrastructures de refroidissement, afin d'être capable de mieux observer les effets du temps pris par les migrations et les actions sur les machines.

$t \in T$: l'intervalle ou timestep t .
$poweroff_h^t$: l'hôte h est en train de s'éteindre au timestep t.
$poweron_h^t$: l'hôte h est en train de s'allumer au timestep t.
$migratefrom_{jh}^t$: la tâche j migre depuis l'hôte h au timestep t
$migrateto_{jh}^t$: la tâche j migre vers l'hôte h au timestep t
p_h^t : l'hôte h est allumé à l'instant t
e_{jh}^t : la tâche j est exécutée sur l'hôte h au timestep t
α_{jh}^t : fraction de CPU alloué à la tâche j sur l'hôte h au timestep t
Les variables α_{jh}^t, e_{jh}^t et p_h^t sont des extensions des variables α_{jh}, e_{jh} et p_h définies en 4.2.1.

$$\forall t \forall h \qquad p_h^t \geq poweroff_h^t \tag{5.1}$$

La contrainte 5.1 définit le fait que si un hôte est éteint, alors il n'est pas en train de s'éteindre.

$$\forall t \forall h \qquad p_h^t \geq poweron_h^t \tag{5.2}$$

La contrainte 5.2 définit de la même manière que précédemment que si un hôte est éteint, alors il n'est pas en train de s'allumer.

$$\forall t \forall h \qquad p_h^t - poweroff_h^t \leq \sum_j e_{jh}^t \tag{5.3}$$

La contrainte 5.3 définit que si aucune tâche n'est sur l'hôte h alors soit celui-ci est allumé et en train de s'éteindre, soit celui-ci est déjà éteint.

$$\forall t \forall h \qquad poweroff_h^t + poweron_h^t \leq 1 \tag{5.4}$$

La contrainte 5.4 décrit le fait qu'un hôte ne peut pas être à la fois en train de s'éteindre et de s'allumer.

$$\forall t \forall j \forall h \qquad 1 - e_{jh}^t \geq poweroff_h^t \tag{5.5}$$

La contrainte 5.5 définit que si au moins une tâche est en train de s'exécuter sur un hôte, alors celui-ci n'est pas en train de s'éteindre.

$$\forall t \forall j \forall h \qquad 1 - e_{jh}^t \geq poweron_h^t \tag{5.6}$$

De la même manière que pour la contrainte 5.5, la contrainte 5.6 définit qu'une tâche qui s'exécute sur un hôte implique que celui-ci n'est pas en train de s'allumer.

$$\forall t \forall h \qquad p_h^t \leq 1 - poweron_h^{t+1} \tag{5.7}$$

La contrainte 5.7 définit que si un hôte était allumé à l'instant t, alors il ne peut pas être en train de s'allumer à l'instant $t+1$.

$$\forall t \forall h \qquad p_h^{t+1} - poweron_h^{t+1} \leq p_h^t \tag{5.8}$$

La contrainte 5.8 décrit que si un hôte était éteint à l'instant t, et qu'il s'allume à $t+1$, h sera en train de s'allumer (et par conséquent allumé) à $t+1$.

$$\forall t \forall j \qquad \sum_h migrefrom_{jh}^t \leq 1 \tag{5.9}$$

$$\forall t \forall j \qquad \sum_h migreto_{jh}^t \leq 1 \tag{5.10}$$

Les contraintes 5.9 et 5.10 définissent qu'une tâche migre depuis et vers au plus un hôte.

$$\forall t \qquad \sum_j \sum_h migrefrom_{jh}^t = \sum_j \sum_h migreto_{jh}^t \tag{5.11}$$

La contrainte 5.11 décrit qu'il y a autant de migration source que de migration destination. Notons que cette contrainte est induite par les contraintes précédentes, et est donc juste présente pour la lisibilité du modèle.

$$\forall t \forall j \qquad \sum_h e_{jh}^t = 1 + \sum_h migrefrom_{jh}^t \tag{5.12}$$

$$\forall t \forall j \qquad \sum_h e_{jh}^t = 1 + \sum_h migreto_{jh}^t \tag{5.13}$$

Les contraintes 5.12 et 5.13 définissent qu'une tâche est sur 1 seul hôte si elle ne migre pas, et sur 2 si elle est en train de migrer. Notons ici que cette contrainte remplace la contrainte 4.7 qui définissait précédemment qu'une tâche était sur un seul hôte.

$$\forall t \forall j \forall h \qquad e_{jh}^t - migrefrom_{jh}^t = e_{jh}^{t+1} - migreto_{jh}^{t+1} \tag{5.14}$$

La contrainte 5.14 définit plusieurs choses. Tout d'abord le fait que si une tâche est sur h à l'instant t, et qu'elle migre depuis h, alors elle n'est plus sur h au timestep $t+1$. Elle définit aussi que si une tâche est sur h à l'instant t et qu'elle ne migre pas, alors elle est toujours sur h au timestep $t+1$. Ensuite la contrainte définit que si une tâche n'est pas sur un hôte h à l'instant t et qu'elle était en train de migrer vers h, alors celle-ci est sur h à l'instant $t+1$. Cette contrainte nous empêche d'avoir à la fois $migrefrom_{jh}^t = 1$ et $e_{jh}^t = 0$, de même que $migreto_{jh}^t = 1$ et $e_{jh}^t = 0$, c'est-à-dire migrer une tâche depuis ou vers un hôte sur lequel elle n'est pas. Enfin, cette contrainte empêche la migration d'une tâche depuis un hôte vers ce même hôte. Bien que cette contrainte ne soit pas nécessaire, elle est un effet collatéral et n'est pas gênante dans la mesure où il n'y a à priori pas de cas où il sera opportun d'effectuer une migration depuis un hôte vers lui même.

$$\forall t \forall h \forall h \neq h' \quad \alpha_{jh}^t + (1 - migratefrom_{jh}^t) \geq \alpha_{jh'}^t - (1 - migrateto_{jh'}^t) \tag{5.15}$$

$$\forall t \forall h \forall h \neq h' \quad \alpha_{jh}^t + (1 - migrateto_{jh}^t) \geq \alpha_{jh'}^t - (1 - migratefrom_{jh'}^t) \tag{5.16}$$

Les contraintes 5.15 et 5.16 définissent respectivement que $\alpha_{jh}^t \geq \alpha_{jh'}^t$ si la tâche j migre depuis l'hôte h vers l'hôte h', et que $\alpha_{jh}^t \geq \alpha_{jh'}^t$ si la tâche j migre vers l'hôte h depuis l'hôte h'. Cela équivaut au final à avoir $\alpha_{jh}^t = \alpha_{jh'}^t$ lorsque l'on a une migration de la tâche j. Autrement dit, le modèle implique que si on a des hôtes homogènes, alors nous voulons avoir la même consommation de ressource à la fois sur l'hôte source et destination.

Il est important de noter que le modèle de consommation des ressources par les tâches qui migrent, défini par les contraintes 5.15 et 5.16, contraint les deux parties d'une tâche qui migre

à consommer le même montant de ressources, comme nous l'avons défini dans notre modèle. Cependant, si il s'avère que ce modèle n'est que peu proche de la réalité, nous pouvons modifier les contraintes de manière à avoir $\alpha_{source} = \eta\alpha_{destination}$.

Afin de définir le fait qu'un hôte qui s'allume ou qui s'éteint consomme des ressources, il faut modifier la consommation d'un hôte à un timestep t comme suit avec l'équation 5.17 :

$$\begin{aligned} \forall h \forall t \quad E_h^t = C_h^{min} p_h^t + (C_h^{max} - C_h^{min}) \sum_j \alpha_{jh} \\ + poweroff_h^t \times (C_h^{max} - C_h^{min}) \\ + poweron_h^t \times (C_h^{max} - C_h^{min}) \end{aligned} \tag{5.17}$$

Logiquement, la consommation instantanée du système au timestep t sera :

$$\forall t \quad E^t = \sum_h E_h^t \tag{5.18}$$

Enfin, la consommation d'énergie du système sur la durée sera la somme des consommations aux instant t.

Le yield minimum ne fait aussi plus sens tel que définit précédemment, il faut donc le modifier. L'équation 5.19 définit le yield minimum à l'instant t comme précédemment pour une tâche qui ne migre pas, et comme la moyenne des yields de la tâche sur l'hôte source et destination si elle migre.

$$\forall j \forall t \qquad \sum_h \frac{\alpha_{jh}^t}{r_{jk} \times (1 + migrefrom_{jh}^t + migreto_{jh}^t)} \geq Y^t \tag{5.19}$$

Si on veut limiter par exemple le nombre de migrations qu'il peut y avoir à un instant t, on peut le contraindre simplement en rajoutant une contrainte telle que la contrainte 5.20, qui définit le nombre maximum de migration à ne pas dépasser à chaque instant t.

$$\forall t \qquad \sum_j \sum_h migrefrom_{jh}^t \leq MAX\_MIGRATIONS \tag{5.20}$$

Il faut enfin ajouter la notion de violation de SLA telle que définie précédemment, c'est-à-dire que lorsqu'une tâche ne se voit pas allouer suffisamment de ressources par rapport à ce qu'elle utilise, elle sera considérée en violation de SLA. Il est important de noter que la violation de SLA définie ainsi est moins continue que le yield. Si le yield dénote un taux de satisfaction, par exemple lorsque l'on alloue 50% ou 60% de ce qu'une tâche avait demandé, la violation de SLA est quand à elle plus binaire, soit il y a violation, soit il n'y a pas violation. Ainsi, par la suite, puisque nous tenterons de minimiser le nombre de violations de SLA, nous pouvons rajouter une telle contrainte dans le modèle. Rappelons d'abord que selon la contrainte 5.21, le yield minimum à l'instant t Y^t doit être supérieur à la borne Y_{bound}^t, afin d'éviter d'avoir une tâche avec une satisfaction proche de 0%, car une violation de 1% par rapport au SLA n'est pas la même chose qu'une tâche arrêtée et ne doit donc pas être traitée comme telle.

$$\forall t \qquad Y_{bound}^t \leq Y^t \tag{5.21}$$

Considérons à présent l'expression de la borne Y_{bound}^t à l'instant t dans l'équation 5.22.

$$\forall t \qquad Y_{bound}^t = Y_{SLA}^t - margin \tag{5.22}$$

La valeur Y_{bound}^t reste donc la borne à ne pas dépasser par les allocations, une limite dure. L'introduction de la valeur Y_{SLA}^t, qui représente le SLA, une valeur que l'on peut dépasser,

autrement dit une limite molle, mais en dessous de laquelle l'allocation de la tâche sera considérée en violation. Enfin, $margin$ représente à quel point une tâche peut être en violation. Prenons un exemple avec des valeurs de $Y^t_{SLA} = 70\%$, avec $margin = 20\%$. Nous aurons donc une valeur de $Y^t_{bound} = 50\%$, ce qui veut dire que nous ne pourrons allouer une tâche en dessous de 50% de ce qu'elle avait requis, mais qu'une allocation entre 50% et 70% sera considérée comme une violation de SLA.

Soit une nouvelle variable binaire v^t_j qui représente une violation de la tâche j à l'instant t.

$$\forall t \forall j \qquad Y^t_{SLA} - Y^t_j < v_j \tag{5.23}$$

La contrainte 5.23 définit la violation de SLA, c'est-a-dire que lorsque le yield de la tâche j Y^t_j est inférieur à Y^t_{SLA}, la tâche j est en violation de SLA.

Nous pouvons donc ajouter une contrainte sur le nombre maximum de violations $MAX\_VIOLATIONS$ de SLA que l'on peut avoir dans une allocation à chaque timestep avec la contrainte 5.24.

$$\forall t \qquad \sum_j v_j <= MAX\_VIOLATIONS \tag{5.24}$$

Pour conclure, il faut noter que, du fait du fonctionnement des solveurs vis-à-vis des contraintes, une solution optimale calculée à partir de cet ensemble de contraintes ne sera optimale que par rapport à cet ensemble de contrainte. Il est donc difficile de comparer les solutions optimales et approchées calculées par les différents algorithmes. En effet, si l'on prend un exemple avec $MAX\_VIOLATIONS = 100$, la solution optimale en énergie sera avec exactement un nombre de violations égal à 100, qui sera en fait une solution optimale avec exactement ce nombre de violations, et non pas une solution optimale en énergie, ce qui pourra différer des algorithmes approximés qui auront eux par exemple moins de violations mais une énergie consommée plus grande. De la même manière, si l'on avait changé l'objectif à minimiser le nombre de violations, la solution optimale n'aurait pas tenu compte de l'énergie et aurait donné une solution non énergétiquement efficace, mais sans violation. Notons au passage que nous observerons un effet similaire avec la contrainte 5.20 sur le nombre de migrations.

Le calcul de l'optimal n'est donc pas cohérent avec la manière dont va s'utiliser l'infrastructure de simulation définie en 5.3, qui correspond au cas où $Y^{bound} = Y^t_{SLA}$ et $margin = 0$.

C'est pourquoi nous n'allons pas utiliser de calcul de résultats optimaux dans les différentes expériences. Ce modèle restera cependant une modélisation de la réalité, et nous pourrons de plus nous en servir comme validation des allocations, c'est-à-dire à savoir si une allocation sera valide car elle se conforme à chacune des contraintes, et ainsi détecter rapidement l'éventualité d'une allocation défectueuse.

5.3 Simulations

Maintenant que le modèle est défini, nous pouvons à présent utiliser ce modèle pour en étudier les impacts. Pour ce faire, nous allons définir l'environnement de simulation. Les travaux, présentés dans [25] et que nous présentons ici ont été effectués conjointement avec Michael Maurer et Ivona Brandić de la Vienna University of Technology, dans le cadre de l'action COST IC0804 [4].

5.3.1 Environnement de simulation

Pour définir l'environnement de simulation, il faut commencer par définir où nous nous plaçons dans la boucle autonomique MAPE-K définie précédemment. Puisque nous nous concentrons sur

Service Level Agreement (SLA)		
CPU	≥ 1024	MIPS
MEM	≥ 512	MB
Storage	≥ 2	GB
TX	≥ 5	MBps
RX	≥ 10	MBps

TABLE 5.1 – Définition d'un SLA

la partie allocation et réallocation de ressources, notre approche se situera dans les parties analyse et planification. Nous supposerons donc que la partie exécution et monitoring seront gérées dans le simulateur, mais ne seront pas étudiées.

Nous devons donc dans les parties analyse et planification définir quelles seront les actions à prendre pour être efficaces à la fois du côté de la qualité de service, ici modélisée par des SLA qui seront décrits ci-après, et à la fois le côté consommation d'énergie de l'infrastructure.

Concrètement, nous allons définir des actions réactives et proactives pour atteindre ces buts.

Reconfiguration de machines virtuelles

La première action sera la reconfiguration des machines virtuelles. Elle tentera de provisionner au plus proche des besoins des tâches pour réduire autant que possible le gaspillage de ressources dû à une surprovision. Les provisions ainsi définies seront utilisées ensuite par l'algorithme de réallocation comme les bornes sur la performance définies précédemment : Y^t_{bound}.

Cette reconfiguration devra aussi s'acquitter d'autres objectifs qui seront de minimiser les violations de SLA. Si nous cherchons à réduire l'intervalle entre ce qui est attribué à une tâche et ce qu'elle avait demandé, il apparait que des variations brusques dans les demandes entraineront une violation de SLA. Nous avons donc deux objectifs antagonistes : réduire le gaspillage dans les ressources attribuées, et avoir de la marge de manoeuvre pour éviter les violations de SLA. Un autre objectif lié à la réduction ou non des ressources allouées à une tâche sera de minimiser le nombre de réallocation de machines virtuelles. En effet, si le système est relativement stable, c'est-à-dire si nous ne reconfigurons pas souvent les machines virtuelles, nous n'aurons pas besoin de bouger les machines virtuelles.

Il faut alors se poser la question de comment est définie une violation de SLA. Comme introduit dans [79], la distinction est faite entre ce qui est demandé par la tâche, ce qui est fourni au service par le système, et ce que le service consomme réellement. Ce qui est demandé par la tâche, le SLA lui même est comme le montre le tableau 5.1 un ensemble de contraintes sur les différentes ressources proposées dans la définition des ressources d'une machine virtuelle. Ce qui est attribué à la tâche correspond à ce qui sera effectivement alloué la tâche par l'algorithme de décision du gestionnaire de tâches, soit α^t_{jh}. Cette valeur sera bien sûr supérieure ou égale à ce que le service va effectivement consommer. Enfin, ce que la tâche consomme réellement est une portion de ce qu'elle avait demandé. Une tâche peut demander par exemple 1000MIPS mais n'en consommer la majorité du temps que la moitié. Cette valeur devra donc toujours être inférieure ou égale à ce que la tâche avait demandé (Y^t_{SLA}).

Nous pouvons à présent définir ce que nous appelons une violation de SLA. Une violation de SLA aura lieu lorsque la tâche tentera de consommer plus de ressources que ce qui lui a été attribué. Cela veut dire que si l'on alloue à la tâche moins que ce qu'il avait demandé, celui-ci ne sera en violation de SLA seulement si il tente de consommer plus que son allocation. Prenons un exemple. Nous avons une tâche qui demande 1000 MIPS de CPU. À un instant donné, est alloué à celle-ci 1200 MIPS. Ici, nous ne pourrons pas avoir de violation de SLA, puisque même si la VM

consomme son maximum, c'est-à-dire 1000 MIPS, ce sera toujours moins que les 1200 alloués. Si par contre à un autre moment, la VM se voit alloué 800 MIPS mais tente d'en consommer 900, il y aura à ce moment là une violation de SLA. Enfin, si nous allouons 600 MIPS à cette même machine virtuelle, mais qu'elle n'a besoin que de 500, nous n'aurons pas de violation de SLA.

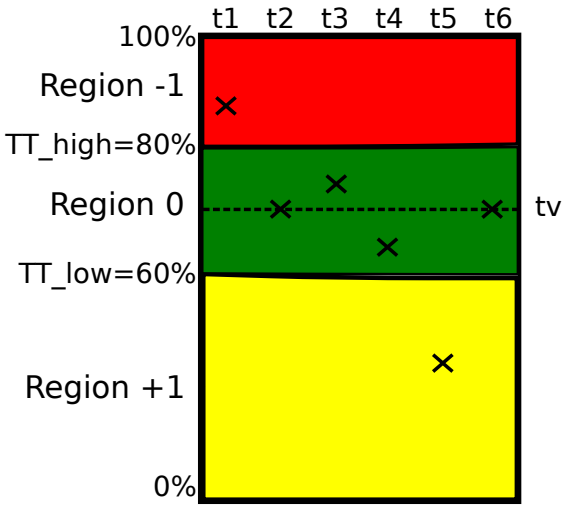


FIGURE 5.6 – Principe des seuils pour l'approche par règles

Afin de gérer proactivement le provisionnement des machines virtuelles sur les machines physiques, nous utiliserons une approche basée sur des règles et des seuils de menace (ou *threat thresholds, TT*). Nous définirons pour chaque SLA deux TTs, un TT haut et un TT bas. Ceux-ci serviront à définir l'orientation que doit prendre le provisionning des machines virtuelles. Comme l'illustre la figure 5.6, si nous avons une utilisation des ressources supérieure au TT haut, la ressource en question est sous-provisionnée, et donc peut tendre à manquer. Si au contraire, nous avons une ressource utilisée en dessous du TT bas, c'est l'effet inverse, c'est-à-dire que trop de ressources ont été allouées par rapport à ce qu'elle consomme. Entre les deux seuils, la ressource est considérée comme bien provisionnée.

Pour chacune des deux régions $+1$ et -1 sont définies des règles de transformation : une pour augmenter l'allocation de ressource dans le cas $+1$, et une pour la diminuer dans l'autre cas. Ces règles sont définies comme suit :

1 **IF**
2 $utilisation^r > TT^r_{\text{high}}$ AND $utilisation^r_{prdite} > TT^r_{\text{high}}$
3 **THEN**
4 Affecter $provision^r$ à $\frac{\text{utilisé}^r}{valeur\_cible(ressource)}$ pour les politiques avec beaucoup de ressources disponibles.
5 Affecter $provision^r$ à $\min(\frac{\text{utilisé}^r}{valeur_cible(r)}, SLO^r * (1+\epsilon/100))$ pour les politiques avec peu de ressources disponibles.

FIGURE 5.7 – Schéma de la règle pour augmenter la provision d'une ressource r

1 **IF**
2 $utilisation^r < TT^r_{\text{low}}$ AND $utilisation^r_{prdite} < TT^r_{\text{low}}$
3 **THEN**
4 Affecter $provision^r$ à $\max(\frac{\text{utilisé}^r}{valeur\_cible(r)}, provision\_minimale^r)$.

FIGURE 5.8 – Schéma de la règle pour diminuer la provision d'une ressource r

Gestion des machines physiques

La seconde action proactive et réactive que le gestionnaire de tâches va entreprendre est celle de la gestion des machines physiques, c'est-à-dire la gestion de l'allumage et de l'extinction des hôtes. Pour ce faire, il faut tenter de répondre à la question : combien pouvons nous éteindre de machines physiques sans pour autant impacter les SLA des tâches. Ainsi, nous avons implémenté une stratégie d'extinction des hôtes permettant itérativement d'éteindre un certain nombre d'hôtes parmi les machines physiques que l'on peut éteindre, c'est-à-dire celles qui peuvent être vides (i.e : qui n'ont pas de machines virtuelles allouées).

À un instant donné, nous déciderons d'éteindre une portion de ces hôtes vides telle que :

$$\text{Nombre de machines physique à éteindre} = \frac{\text{Nombre de machines physique vide}}{a}$$

Ainsi, lorsque le nombre de machine physique à éteindre restera constant, cette formule permettra d'éteindre toutes les machines physiques sauf une, qui restera pour faire tampon en cas de pic de charge. Cette technique permettra d'arrêter toutes ces machines sauf une en $t = \lceil \log_a \frac{1}{n} \rceil$ timesteps. Cela permettra d'éteindre des machines en peu de temps, mais de manière à en garder certaines en cas de variation brusque des demandes du système.

Pour l'allumage des machines, nous fonctionnerons de manière similaire à la gestion de la reconfiguration avec des règles et des seuils. Nous surveillons ainsi les valeurs moyennes d'utilisations de chaque ressources. Si au moins une dépasse un seuil ($TT^{ressource}_{PM}$), nous allumerons des machines autant de machines qu'il faudra pour faire redescendre la moyenne en dessous du seuil correspondant. Par exemple, si l'utilisation moyenne de CPU dépasse notre seuil de 80% pour arriver jusqu'à 90% il faudra allumer autant d'hôtes que nécessaire pour que la nouvelle moyenne redescende en dessous de 80%. Nous choisirons bien évidemment les hôtes à allumer les plus efficaces énergétiquement parlant.

Réallocation des machines virtuelles

La dernière action que nous pouvons prendre est celle de la réallocation des machines virtuelles. Si nous faisons une allocation initiale et que les besoins des tâches changent, il faut réallouer les machines potentiellement différemment. Nous avons donc implémenté différents algorithmes de réallocation de machines virtuelles. Ces algorithmes calculent un ensemble de migration pour passer d'un état donné vers un autre en tentant d'économiser de l'énergie.

FIRSTFIT Le premier algorithme que nous avons développé est un algorithme basé sur le principe du FIRSTFIT. À la différence du FIRSTFIT classique, cet algorithme doit tenter de faire une réallocation de machines virtuelles, et ainsi ne peut pas simplement faire une allocation des tâches sur les premiers hôtes. Il faut donc l'adapter de manière à conserver un comportement similaire au FIRSTFIT, mais dans ce nouveau contexte.

La première allocation se fera comme décrit dans le chapitre 4. Pour les réallocations qui suivront, l'allocation se fera en deux étapes. La première sera de trouver l'hôte le plus chargé

parmi les hôtes non vides, pour ensuite distribuer une certaine portion de sa charge à la manière d'un algorithme FirstFit, c'est-à-dire pour chaque tâches sur le premier hôte sur lequel il rentre. La seconde étape consistera à prendre l'hôte le moins chargé qui n'est pas destination d'une migration de tâche, et de distribuer la charge de cet hôte tâche de manière FirstFit.

RoundRobin Le second algorithme développé est basé sur le comportement de l'algorithme RoundRobin décrit précédemment. La première allocation se fera de la même manière. Pour les réallocations, de la même manière que pour le first fit, nous prenons l'hôte le plus chargé pour en distribuer la charge à la manière d'un RoundRobin. Ensuite, une machine virtuelle sera choisie sur chaque hôtes pour migrer vers une machine physique vide. Le but étant ici d'avoir à la fois une composante de consolidation et d'équilibrage de charge. Cet algorithme n'aura pas de bonnes performances en terme de consommation d'énergie, mais il nous servira de comparaison pour les autres algorithmes, puisqu'il aura un comportement facilement prédictible.

C'est finalement l'avantage des deux algorithmes FirstFit et RoundRobin, qui sont des algorithmes bien connus, de pouvoir conserver un comportement que l'on connait, prédictible et simple, afin de pouvoir avoir une base de comparaison sur un comportement spécifique, la consolidation pour le FirstFit et l'équilibrage pour le RoundRobin.

MonteCarlo Nous avons ensuite développé un algorithme basé sur le fonctionnement de la méthode de Monte Carlo, qui est une technique probabiliste utilisée pour calculer des valeurs numériques. Dans notre cas, nous effectuerons une première allocation en RoundRobin, afin d'avoir un système équilibré, ce qui permettra de converger plus vite vers une bonne solution. Pour les réallocations, l'algorithme calcule la valeur de l'allocation courante, c'est-à-dire une valeur basée sur un certain nombre de paramètres qui en augmentent ou diminuent la valeur. Ces paramètres visent à effectuer les modifications de valeur suivantes :

- Augmenter le coût pour chaque tâche qui migre.
- Augmenter le coût pour chaque hôte considéré comme surchargé.
- Diminuer le coût pour chaque hôte qui est vide, ou qui sera vide au timestep suivant.

Le poids de chaque paramètre peut être modifié pour changer son impact. Dans nos expériences, nous avons choisi des valeurs de 1 pour les tâches qui migrent, 4 pour les hôtes surchargés, -10 pour les machines physiques qui sont ou seront vides, et enfin 90% pour une machine considérée comme étant surchargée.

Cela veut dire que par exemple, si nous avons une instance de 3 hôtes, avec une machine virtuelle consommant 50% de CPU migrant depuis l'hôte 1 vers l'hôte 2, la valeur de l'instance sera de $1 \times 1 + 4 \times 0 + (-10 \times 2) = -19$. L'algorithme calcule ensuite différentes réallocations choisies aléatoirement par un ensemble de migrations. Si la valeur d'une allocation est inférieure à celle précédente, nous gardons cette nouvelle allocation. Cette étape est répétée un nombre défini de fois, 100 fois dans notre cas, afin de faire émerger le meilleur ensemble de migration à effectuer pour la prochaine itération.

Nous avons choisi d'implémenter un algorithme suivant une méthode de Monte Carlo, car cela nous permettait de pouvoir donner des poids différents à certaines actions comme la migration ou l'allumage d'un hôte. Ainsi, nous pouvons modifier facilement l'algorithme pour pouvoir converger vers un certain type de solutions, comme une solution avec peu de migration si les migrations ont un fort poids.

VectorPacking Enfin, le dernier algorithme défini est l'algorithme basé sur le VectorPacking comme décrit précédemment en section 4.3. Pour la première allocation, nous effectuons une allocation en utilisons l'algorithme VectorPacking décrit en 4.3. Pour les réallocations ensuite, l'algorithme va tenter de consolider autant que possible en utilisant la même heuristique que pour

l'allocation initiale. Pour cela, on va effectuer un ensemble de migrations en partant des tâches les plus volumineuses, séparées en deux listes de manière à contrebalancer le déséquilibre sur les hôtes destination. Ensuite, il y aura une étape d'équilibrage depuis les machines physiques les plus chargées vers celles qui le sont le moins et qui ne seront pas vides dans l'itération suivante. Ainsi, l'algorithme tente de consolider les VM dans le plus petit sous ensemble de machines physiques, pour ensuite équilibrer la charge parmi les machines choisies. Cet algorithme a montré dans les expériences précédentes sur l'allocation statique (cf : section 4.4.2) sec ses bons résultats et son efficacité en terme de consommation d'énergie, c'est pourquoi nous avons conservé un algorithme au comportement similaire.

5.4 Evaluation et expérimentations

Maintenant que nous avons défini l'environnement de simulation utilisé, nous pouvons passer à l'évaluation du système et des effets des différents aspects de celui-ci sur la qualité de service et la consommation d'énergie. Pour cela, il faut commencer par décrire la méthodologie utilisée pour les expériences, comme nous allons le faire dans la suite.

5.4.1 Méthodologie

Nous avons séparé les expériences en 4 expériences distinctes. La première, décrite en section 5.4.2 va nous servir à déterminer le gain potentiel que seules les techniques de reconfiguration de machines virtuelles peuvent apporter. La seconde, en partie 5.4.3 sera orientée sur la performance en terme de consommation des différents algorithmes décrits ci-dessus. Ensuite, la troisième expérimentation nous servira à étudier de plus près l'impact des différents paramètres de l'infrastructure de test pour seulement les deux meilleurs algorithmes issus de l'expérience précédente. Enfin, la dernière expérimentation en 5.4.5 visera à évaluer le passage à l'échelle de tels algorithmes, afin d'étudier si de tels algorithmes peuvent réellement être utilisés dans un système large échelle réel.

Pour toutes les expériences, nous avons choisi de simuler 100 machines physiques avec une capacité de 1.1GHz de CPU et 4GB de mémoire vive. Chaque hôte consomme entre 20W à vide (C^{min}) et 100W lorsqu'ils sont totalement chargés. Nous avons aussi défini 100 machines virtuelles à l'intérieur du système, chacune demandant à $t = 0$, 500MHz de CPU et 500MB de mémoire. Nous avons défini autant de machines virtuelles que de machines physique pour deux raisons : la première étant pour mieux avoir l'effet de la gestion des machines physiques, puisque l'on va en éteindre un certain nombre, la seconde étant pour nous donner plus de marge de manoeuvre, et être dans le cas où l'infrastructure est construite afin de résister aux pics de charges. Pour toutes ces machines virtuelles, nous avons utilisé différents workloads, deux synthétiques et un basé sur des mesures réelles d'une application de bioinformatique présentée en [41] (qui sera abbrévié BOKU par la suite). Pour les workloads synthétiques, la distinction est faite au niveau de leur volatilité, c'est-à-dire à quel point ils changent vite. Le workload LIGHT à une volatilité faible, jusqu'à 10% de changement d'un timestep sur l'autre. Le workload MEDIUM_HEAVY, qui varie beaucoup plus, peut varier jusqu'à 50% entre les différentes itérations de la boucle autonomique. Les détails de la génération de ces différents workloads peuvent être trouvés en [80]. Un exemple peut être trouvé en figure 5.9. Nous avons évalué les différents algorithmes sur 100 itérations, et nous avons choisi une valeur de $a = 2$ pour la stratégie d'extinction des hôtes, c'est-à-dire que nous allons éteindre la moitié des hôtes vides à chaque timestep.

Enfin, pour pouvoir étudier les différents effets avec précision et par souci de simplification, nous ne travaillerons ici que sur un seul cluster homogène (même si le modèle et l'infrastructure de simulation nous permettrai d'avoir de l'hétérogénéité), sans équipement de refroidissement.

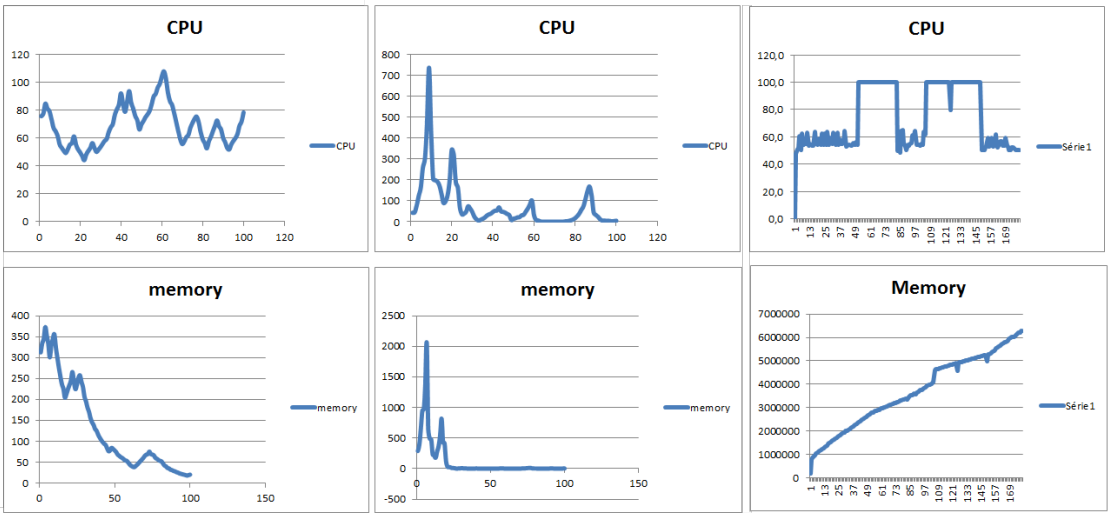


FIGURE 5.9 – Exemple de workloads : LIGHT (à gauche), MEDIUM_HEAVY (au milieu) et BOKU (à droite)

5.4.2 Impact de la reconfiguration de machines virtuelles

Les premières expériences portent sur l'effet de la reconfiguration des machines virtuelles seule sur la consommation d'énergie du système. Pour ce faire, nous avons exécuté les quatres différents algorithmes avec une seule volatilité de workloads (ici nous avons choisi MEDIUM_HEAVY pour sa volatilité plus grande), et un ensemble fixe de paires de seuils de menace. Les différents paramètres de l'évaluation peuvent être retrouvés dans le tableau 5.2.

Paramètre	Valeurs dans l'évaluation
Workloads	MEDIUM_HEAVY volatility
Reconfiguration de VM	allumée/éteinte
TTs pour la reconfiguration de VM	[20%, 40%]
Algorithmes de réallocation	RoundRobin, FirstFit, MonteCarlo, VectorPacking
tt_{PM}^{cpu}	0.8
tt_{PM}^{memory}	0.8

TABLE 5.2 – Différents paramètres à évaluer pour l'expérience 1

La figure 5.10 montre l'énergie totale consommée sur la durée des 100 timesteps. Nous ne montrons ici qu'un seul des résultats sans reconfiguration de machines virtuelles puisque chacun des 4 résultats avec les différents algorithmes ont donné la même consommation d'énergie. En effet, chacun des algorithmes nous donne l'optimal de 2 machines virtuelles par machines physique, du fait des besoins initiaux de 500MHz et 500GB, et puisque nous n'avons pas de reconfiguration de VM ce qui est alloué ne change pas au fur et à mesure de l'expérience. Par contre, si l'on regarde la différence entre avec et sans reconfiguration pour l'algorithme le plus performant (FirstFit), la différence s'accentue jusqu'à 61.6% de différence. Cela est dû à la faible valeur de

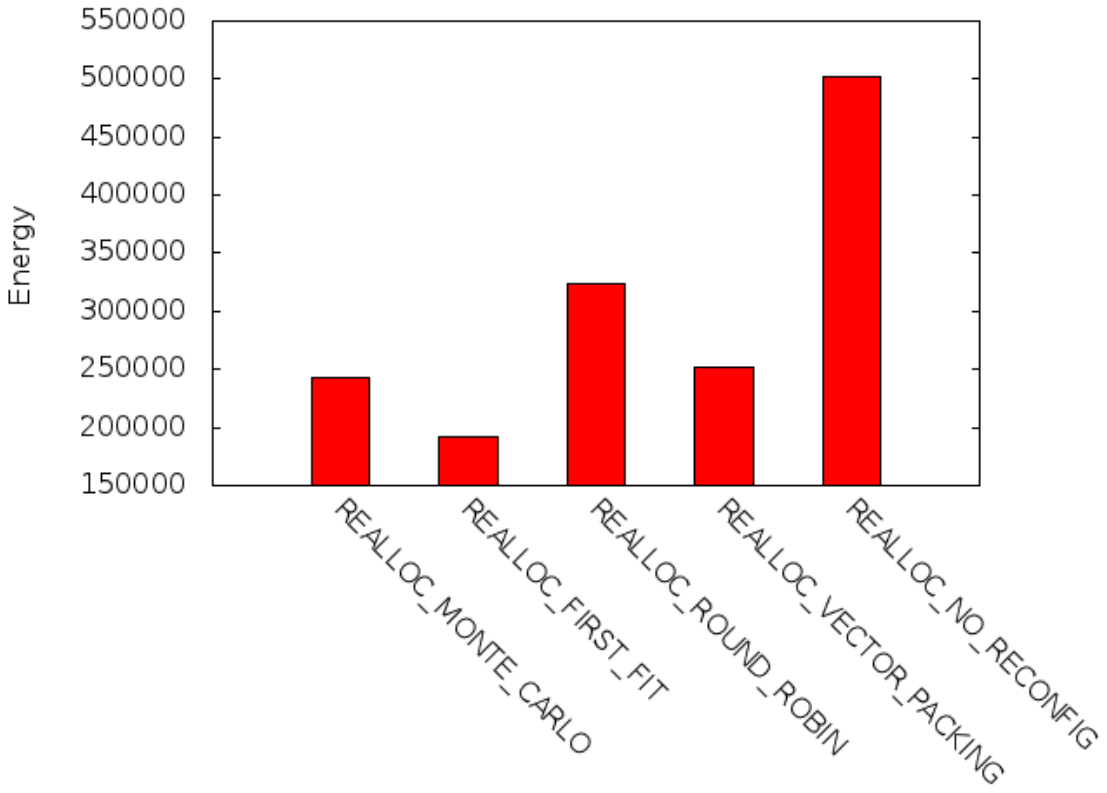


FIGURE 5.10 – Consommation d'énergie pour l'expérience 1

C^{min} par rapport à C^{max} dans nos expériences, ce qui permet de meilleures performances pour les machines sous utilisées, qui ne consomment réellement que peu par rapport à leur besoins. Cela est dû au modèle de consommation d'énergie : si seulement une portion faible vient du fait que la machine soit allumée, la majeure partie vient de l'utilisation induite par les tâches, ce qui implique qu'une machine peu chargée sera encore relativement efficace.

Cependant, ce gain de consommation d'énergie n'est pas sans coût en terme de violation de SLA, comme le montre la figure 5.11. Ces résultats en terme de SLA sont dûs au fait notamment que les algorithmes de réallocation s'attendent à une reconfiguration du système, mais celui-ci varie trop vite, ce qui amène des situations où les tâches sont trop consolidées, et ne peuvent donc pas migrer à temps. Cependant, on remarquera que l'algorithme FIRSTFIT, qui était le plus performant en énergie, est ici le moins performant en terme de violations de SLA.

5.4.3 Évaluation des algorithmes de réallocation

Afin de pouvoir évaluer la performance des algorithmes de réallocation, nous avons exécuté les 4 algorithmes cette fois-ci avec la reconfiguration de machines virtuelles. Cependant, nous avons comme précédemment utilisé une seule paire de seuils de menace. Nous avons évalué les algorithmes sur 3 workloads différents, une faible(LIGHT) et moyenne-haute (MEDIUM_HEAVY) volatilité dans les besoins des VMs, et le workload basé sur des mesures d'application bioinfor-

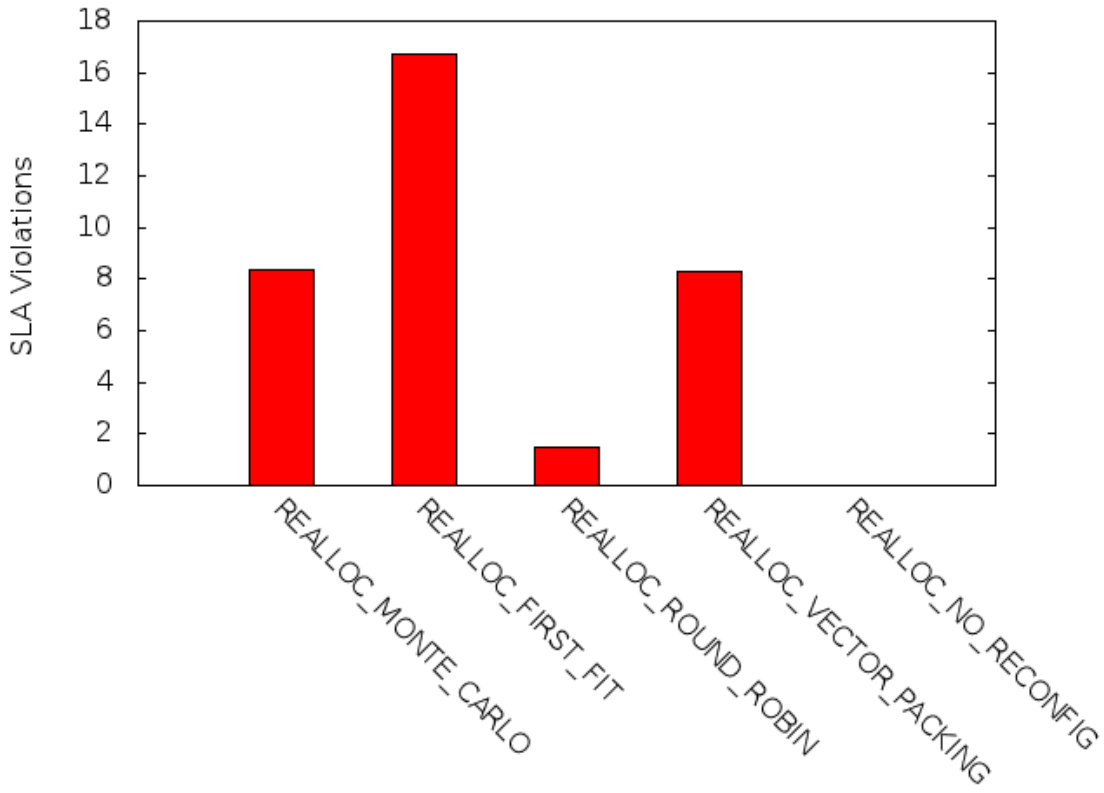


FIGURE 5.11 – Pourcentage de violations de SLA pour l'expérience 1

matique (BOKU). Les différents paramètres sont montrés dans le tableau 5.3.

Paramètres	Valeurs dans l'évaluation
Workloads testés	LIGHT, MEDIUM_HEAVY, BOKU
Reconfiguration de VM	Oui
TTs pour la reconfiguration de VM	[20%, 40%]
Algorithmes de réallocation	RoundRobin, FirstFit, MonteCarlo, VectorPacking
tt_{cpu}	0.8
tt_{memory}	0.8

TABLE 5.3 – Différents paramètres d'entrée pour l'évaluation de la seconde expérience

La figure 5.12 nous montre l'énergie totale consommée par les machines sur les 100 timesteps de l'expérience pour les trois différents workloads. Comme le montre cette figure, pour le workload de faible volatilité, l'algorithme MonteCarlo donne les meilleures performances, suivi de près par l'algorithme VectorPacking et FirstFit. L'algorithme RoundRobin comme nous pouvions nous y attendre n'a pas de bonnes performances en terme de consommation d'énergie, puisque celui-ci consomme environ deux fois plus d'énergie en répartissant fortement les machines virtuelles sur les machines physiques.

Si l'on s'intéresse à un workload plus volatile, MEDIUM_HEAVY, nous pouvons voir que

l'algorithme FirstFit donne les meilleurs résultats, allant jusqu'à 37% de consommation d'énergie en moins per rapport au RoundRobin. Enfin, pour le workload BOKU, qui demande beaucoup plus de ressources, notamment CPU, que les autres workloads, les résultats sont similaire au workload LIGHT, seulement avec une consommation d'énergie globalement plus haute.

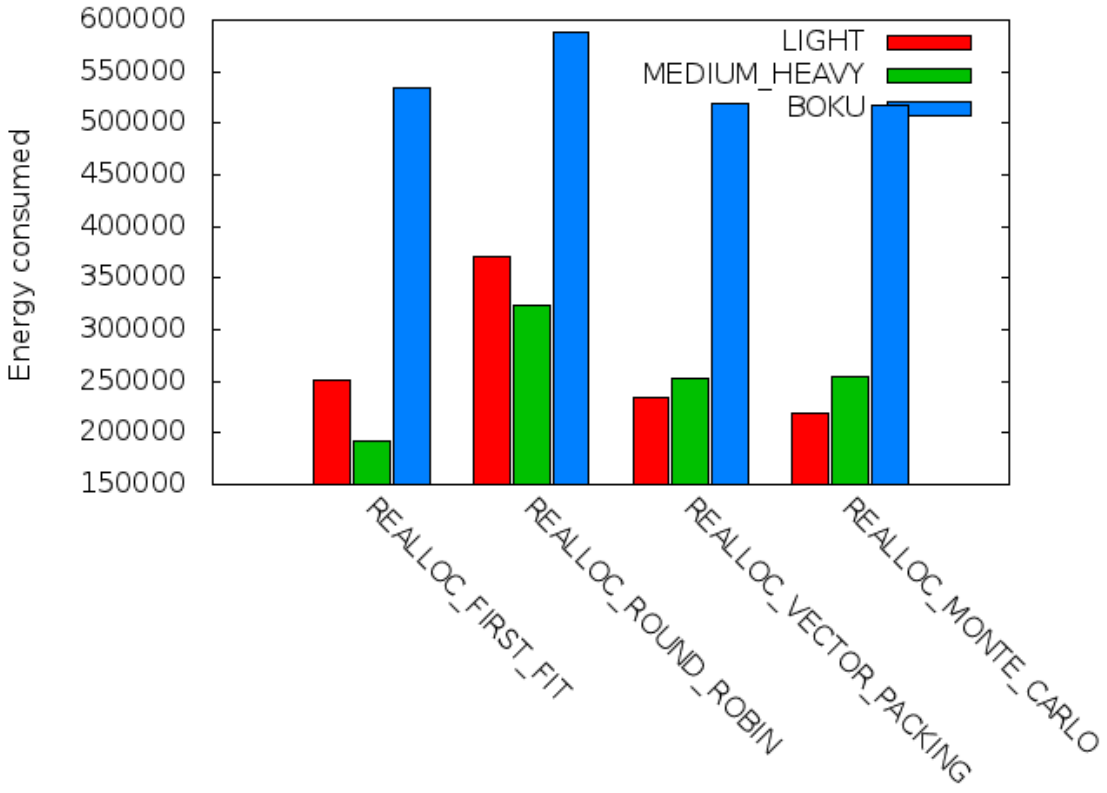


Figure 5.12 – Consommation d'énergie des algorithmes en fonction des différents workloads

La raison derrière ces différences est montrée en partie par la figure 5.14, qui montre le nombre d'hôtes moyens allumés lors de l'expérience. Comme nous pouvons le voir, l'algorithme RoundRobin va tenter d'équilibrer les machines virtuelles sur chaque machine physique, empêchant ainsi le gestionnaire autonomique d'éteindre les hôtes vides. L'algorithme qui donne les meilleurs résultats, VectorPacking, est fait pour tenter de consolider fortement les machines virtuelles, tout en équilibrant si possible sur les machines physiques qui restent allumées. Il faut noter cependant qu'excepté pour le FirstFit et le RoundRobin, le nombre de machines physique allumées augmente au fur et à mesure que les besoins des VM deviennent plus volatiles. Cela peut s'expliquer par le fait que l'algorithme FirstFit est moins proactif que les autres, nous amenant aux problèmes que nous allons voir par la suite dans la figure 5.13.

La figure 5.13 montre le pourcentage de violation pour l'ensemble de l'infrastructure cloud pour chaque algorithme. Seuls les workloads LIGHT et MEDIUM_HIGH sont représentés, puisque le workload BOKU est peu volatile, et n'a pas donné de violation.

Comme nous pouvons le voir, l'algorithme RoundRobin a le moins de violations de SLA, puisqu'il utilise toutes les machines physiques. Ainsi, on peut en déduire que la majorité des

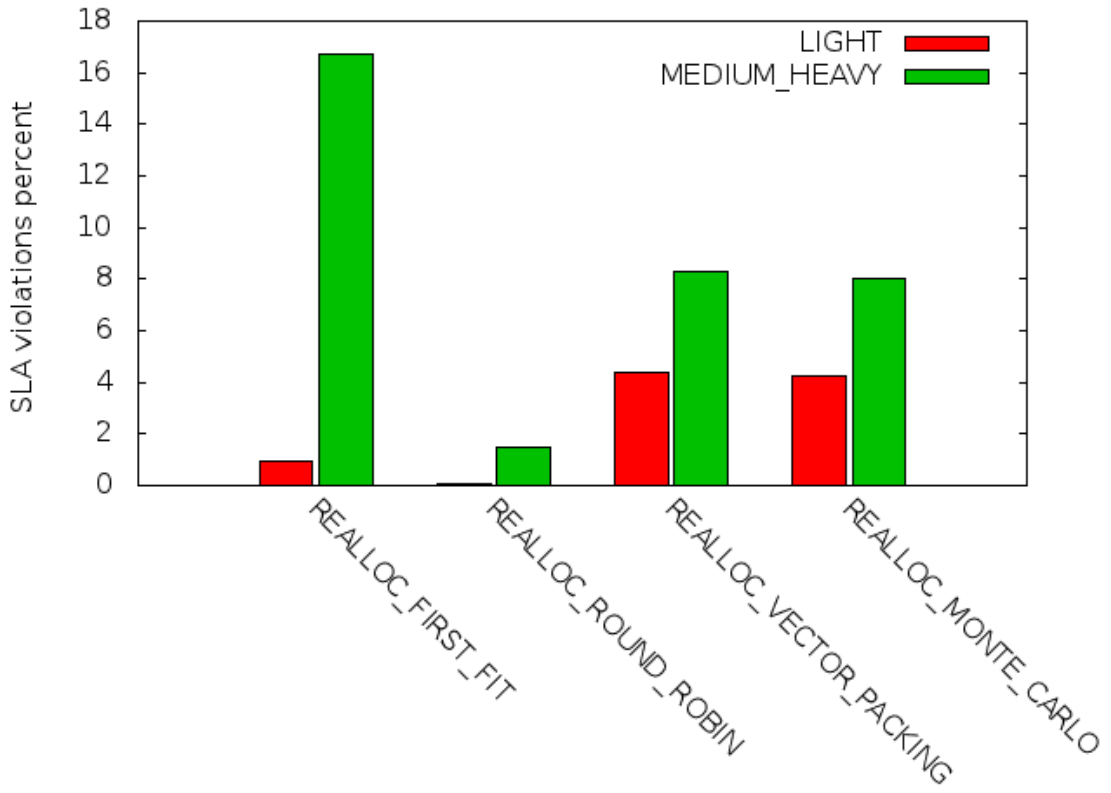


FIGURE 5.13 – Pourcentage de violation de SLA des machines virtuelles en fonction des différents workloads

violations de SLA est générée par la reconfiguration, qui entraine une impossibilité par exemple d'augmenter l'affectation des machines virtuelles. Les algorithmes VECTORPACKING et MONTECARLO ont un nombre de violations de SLA d'environs 4% et 8%. Enfin, l'algorithme FIRSTFIT, qui avait les meilleurs résultats pour le workload à faible volatilité, donne des résultats plus mauvais que les autres puisqu'il arrive jusqu'à 16% de violations de SLA.

Afin de regarder de plus près les performances des algorithmes, il faudra donc prendre en compte à la fois la consommation d'énergie mais aussi les baisses de performances en terme de qualité de service associées, qui seront induites par la reconfiguration de machines virtuelles. L'exemple parfait en est le workload MEDIUM_HEAVY avec l'algorithme FIRSTFIT, qui donne des bons résultats en terme de consommation d'énergie, mais qui amène jusqu'à 16% de violations de SLA. En regardant de manière plus globale, on pourra donc voir qu'on a une différence d'environs 60kW entre les algorithmes MONTECARLO etFIRSTFIT pour environs 8 points de différence en terme de violations de SLA.

5.4.4 Evaluation des seuils de menaces

La prochaine étape dans l'évaluation de nos algorithmes est d'évaluer les différentes paires de seuils que nous utilisons : une qui sert à la gestion des machines physiques (basée sur la consommation moyenne de CPU et d'utilisation mémoire sur l'ensemble des hôtes), que nous appellerons

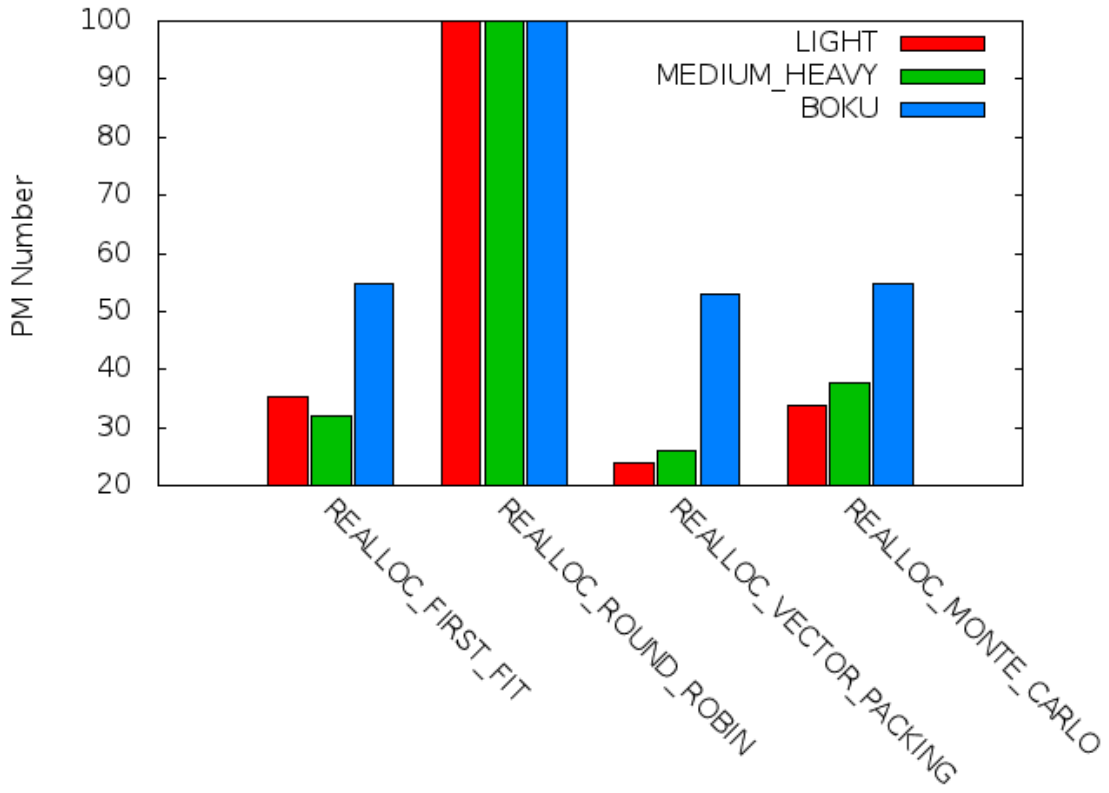


FIGURE 5.14 – Nombre moyen d'hôtes allumés

PM-TTs. L'autre type nous sert à la reconfiguration des machines virtuelles, comme expliqué en 5.3.1, que nous appellerons VM-TTs. Nous évaluerons les deux types en prenant les deux algorithmes qui ont donné les meilleurs résultats dans l'expérimentation d'avant : VECTORPACKING et MONTECARLO, et ce pour le workload MEDIUM_HEAVY.

Numéro de scénario	1	2	3	4	5	6
PM-TTs[CPU,Memory]	[80%, 80%]	[80%, 80%]	[60%, 80%]	[60%, 80%]	[60%, 60%]	[60%, 60%]
VM-TTs[TT_{low}, TT_{high}]	[20%, 40%]	[50%, 75%]	[20%, 40%]	[50%, 75%]	[20%, 40%]	[50%, 75%]

TABLE 5.4 – Scénarios pour l'expérience sur les seuils

Nous allons analyser trois différentes paires de PM-TTs [CPU,MEM] : [80%,80%], [60%,80%] et [60%,60%]. Nous utiliserons des TTs plus prudents pour le CPU dans un des cas, car cette ressource tend à fluctuer plus que la mémoire dans les applications réelles. Cela veut dire que nous voudrons une consommation moyenne des ressources sur un hôte de 60% pour le CPU au lieu de 80%, ce qui aura pour effet, puisque le CPU varie plus vite que la mémoire, de déclencher des allumages et extinctions d'hôtes d'une manière plus conservative. Pour ce qui est des seuils des machines virtuelles, nous utiliserons l'intervalle standard [50%,75%] issue de [80] et qui a prouvé être un bon réglage en général, et qui sera ajouté au VM-TT [20%,40%] utilisé dans l'expérience précédente, qui est quand à lui plus prudent. Cela veut dire que nous mettrons le

seuil haut pour le CPU plus bas que celui de la mémoire. Les VM-TT plus bas auront pour effet de déclencher l'état surchargé plus tôt, et ainsi potentiellement éviter que la consommation des services soit plus haute que la provision, puisque l'état surchargé déclenchera une augmentation de la provision. Un résumé des paramètres peut être trouvé dans le tableau 5.4.

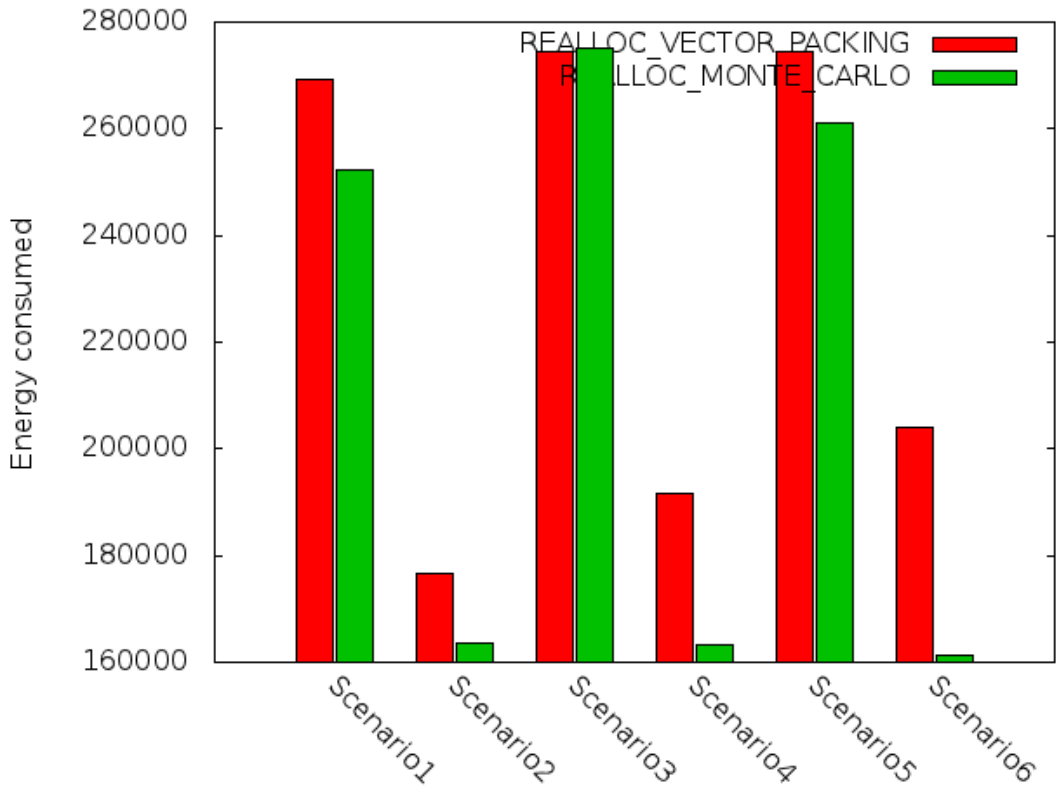


FIGURE 5.15 – Consommation d'énergie pour la variation des différents seuils de VM et de PM

Les figures 5.15 et 5.16 nous montrent que l'algorithme MONTECARLO est quasiment toujours le meilleur en terme à la fois de consommation d'énergie et de violations de SLA. Cependant, celui-ci, comme nous le verrons par la suite, prend trop de temps à calculer les solutions. Pour les scénarios 2, 4 et 6, la différence en faveur de l'algorithme MONTECARLO est très grande pour la consommation d'énergie. Ainsi, sans surprise, les scénarios où la consommation d'énergie est la plus faible sont aussi ceux qui possèdent les plus grands nombres de violations de SLA. Inversement, les scénarios possédant les plus hautes consommations d'énergie ont aussi les plus faibles violations de SLA.

De manière générale, les scénarios numéros pair et impairs nous montrent un comportement similaire, ce qui veut dire que les VM-TTs ont un impact plus grand sur les différents résultats que les PM-TTs. De plus, nous pouvons voir que baisser les valeurs des PM-TTs augmentent la consommation d'énergie sans pour autant avoir un impact favorable sur les violations de SLA dans certains cas.

Enfin, les bons résultats de l'algorithme MONTECARLO peuvent aussi être expliqués en regardant le nombre de machines physiques qui ont été allumées ou éteintes. L'algorithme VEC-

torPacking éteint au moins autant, souvent plus, de machines physiques que l'algorithme MonteCarlo, et dépense ainsi moins d'énergie à éteindre les machines qui ont été allumées inutilement que le VectorPacking.

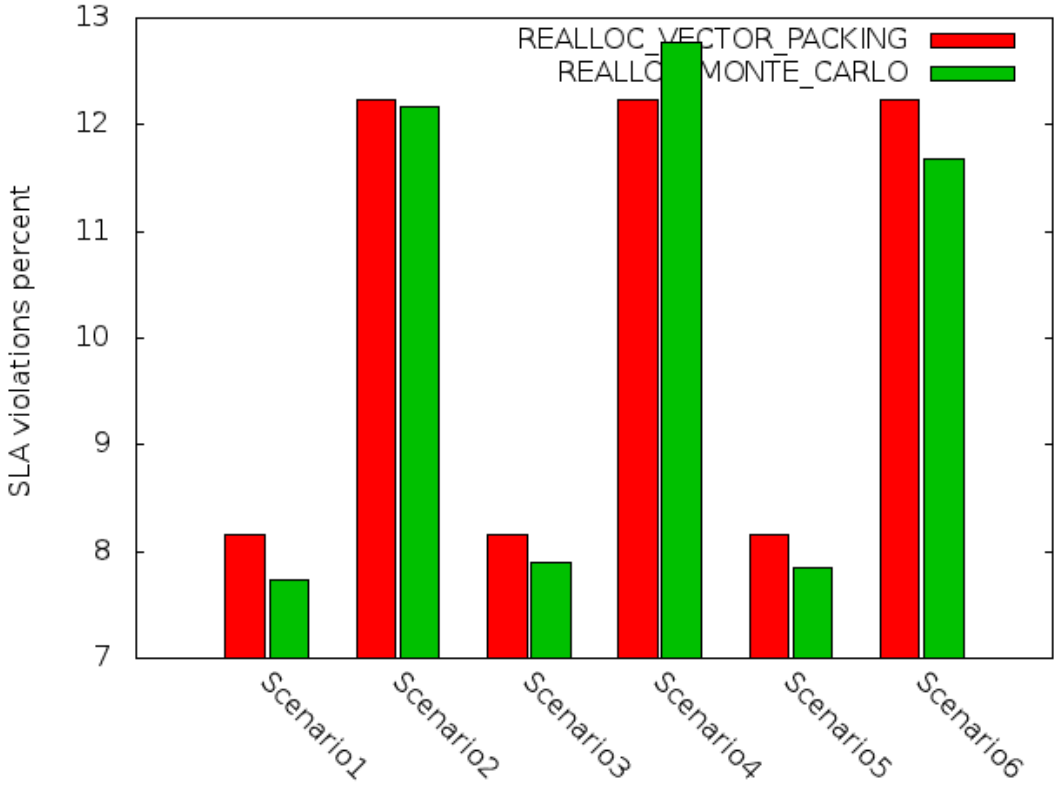


Figure 5.16 – Pourcentage de violations de SLA pour les différents seuils de VM et de PM

5.4.5 Passage à l'échelle

La figure 5.17 montre le temps de calcul des algorithmes de réallocation pour 100, 200, 400 et 800 machines virtuelles. Ces mesures ont été effectuées sur un processeur Intel i7 2600K (un seul coeur a cependant été utilisé). Ces temps d'exécutions comportent à la fois le temps de calcul de la reconfiguration des VMs mais aussi le temps de calcul de la réallocation du système. Comme nous pouvons le voir, l'algorithme MonteCarlo prend environs six fois plus de temps à calculer à chaque timestep pour 100 machines virtuelles, alors que les autres se terminent en un temps raisonnable (de l'ordre de la demi seconde pour 100 machines virtuelles avec le surcoût lié à la reconfiguration). De plus, l'algorithme MonteCarlo passe mal à l'échelle pour deux raisons. La première est qu'il doit calculer un certain nombre de fois la solution (100 fois dans nos expériences), prenant ainsi de plus en plus de temps à calculer l'ensemble des 100 solutions. La seconde raison vient du fait qu'avec l'augmentation du nombre de machines virtuelles et physique, pour arriver à une solution proche optimale à chaque timestep, l'algorithme doit augmenter le nombre d'itérations sur le calcul de solutions intermédiaires. En effet, plus on va augmenter le

nombre de VM et d'hôtes, plus le nombre de combinaisons possible va être grand, et plus la chance de trouver au hasard une bonne solution sera faible, nous amenant à des améliorations de résultats de plus en plus sporadiques. Ainsi, à mesure que la taille des instance augmente, la qualité moyenne de la solution va baisser. Cependant, il faudrait prendre en compte que les algorithmes du type de MonteCarlo peuvent être parallélisés. À l'instar des algorithmes comme FirstFit ou VectorPacking qui convergeront toujours vers la même solution depuis les mêmes paramètres, l'algorithme MonteCarlo qui est basé sur des processus aléatoires donnera des résultats potentiellement différents. Ainsi, on peut facilement paralléliser de tels algorithmes en faisant calculer des solutions sur plusieurs machines, puis récupérer la meilleure. Si une telle solution est choisie, il faudra prendre en compte les ressources consommées par l'algorithme, et donc le surcoût énergétique induit par cette technique, ce qui peut se révéler être inefficace dans beaucoup de cas, puisque l'on a aucune garantie sur la qualité de la solution, ni sur l'amélioration que cela peut apporter.

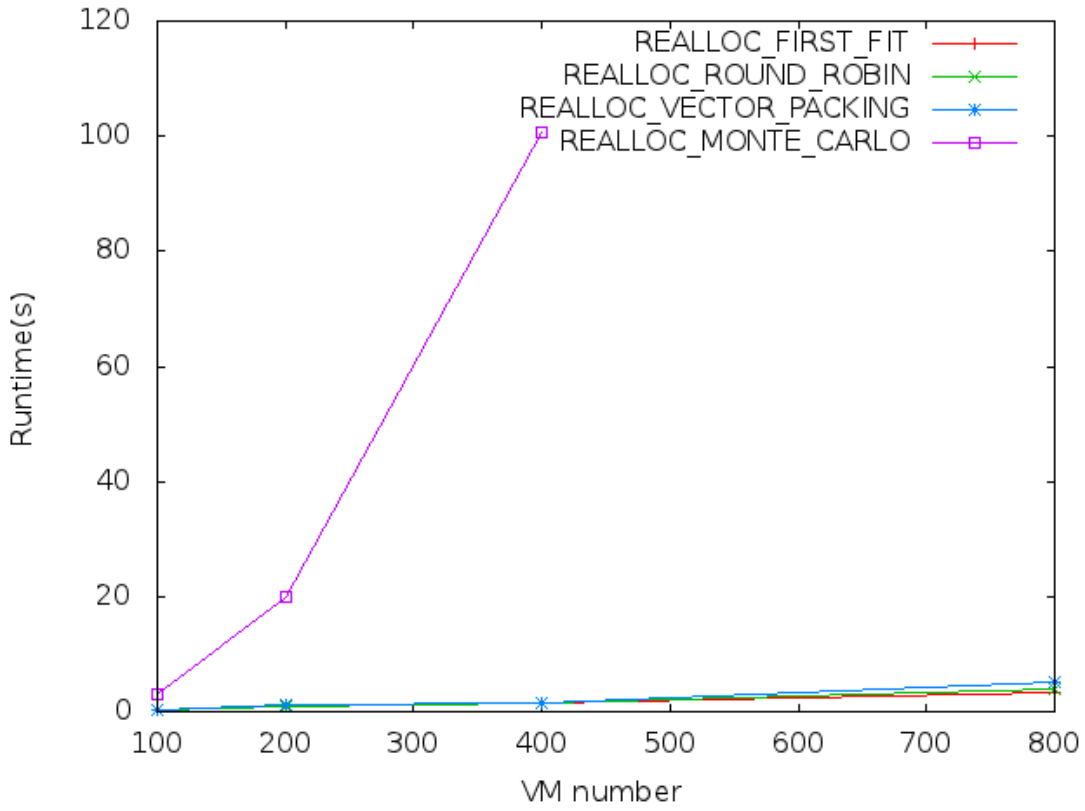


Figure 5.17 – Temps d'exécution des algorithmes pour 100 VMs

En conclusion, la figure 5.17 nous montre que l'algorithme MonteCarlo n'est pas scalable, contrairement aux 3 autres algorithmes dont le temps croit de manière acceptable, avec un temps aux alentours de 5 secondes pour 800 machines virtuelles.

5.4.6 Conclusion, Discussion, Ouverture

Conclusion

Dans cette partie, nous avons décrit les améliorations nécessaires au modèle pour passer d'un modèle statique à un modèle dynamique. Nous avons donc défini un ensemble de contraintes permettant d'étendre le modèle linéaire statique à la dynamicité. Ainsi, nous avons ajouté trois nouveaux aspects : les coûts à la fois en ressource mais aussi en temps de l'allumage des hôtes, de l'extinction des hôtes, et de la migration des machines virtuelles. Les différentes contraintes linéaires de notre nouveau modèle nous ont permi de définir des nouvelles orientations, à la fois au niveau de l'objectif, mais aussi au niveau des algorithmes.

Au niveau des algorithmes, il nous a fallu passer d'un paradigme d'allocation de tâches, à un paradigme de réallocation de tâches. Cela veut dire que nous ne décidons non plus quel service allouer à quel hôte, mais quel service doit migrer, et vers où. Au niveau des objectifs par contre, ceux-ci n'ont pas changé. Même si nous n'avons pas pris en compte le problème du compromis entre l'énergie et la performance, il est apparu que nous pouvions utiliser différents objectifs en plus de l'objectif du problème BoundedYield. Grâce aux contraintes linéaires supplémentaires, nous avons par exemple pu définir plus précisément ce qu'est une violation de SLA, mais aussi eu la possibilité limiter le nombre de migration qu'il doit y avoir pendant un timestep.

Ce faisant, nous avons proposé une infrastructure de gestion basée sur une boucle autonomique permettant d'effectuer autant les reconfigurations de machines virtuelles, que les reconfigurations nécessaire au passage d'un état vers un autre. Ces actions sont faites en vue des changements dans les besoins du système, et en prévision du temps pris par les différentes étapes de migration et par l'allumage et l'extinction des hôtes.

Cette approche multi-niveau nous a permis de séparer les trois parties que sont la reconfiguration de machines virtuelles, la gestion de machines physique et la migration de machines virtuelles. Nous avons montré que la reconfiguration de machines virtuelle seule permet de réduire grandement la consommation d'énergie, en comparaison avec une approche ne tenant pas compte de l'énergie. De plus, si l'on considère les algorithmes qui passent à l'échelle, on peut améliorer les économies d'énergie jusqu'à 37% dans le meilleur cas en gardant un taux de violation nul pour le workload bioinformatique, moins de 4% pour le workload synthétique à faible volatilité, et moins de 8% pour le workload synthétique à plus grande volatilité.

Discussion

En pratique, les différentes expériences nous ont montrés que les mêmes recommandations pour le choix de l'ordre d'allocation sur les hôtes tenaient toujours, à savoir affecter sur les hôtes les moins consommateurs au maximum. Par contre, vis-à-vis de la métrique de violation de SLA, il apparaît plus propice de tenter de bouger les tâches les plus petites en premier. En effet, du fait du surcoôt dû à la migration, il y aura moins de surcoût induit, par conséquent le système amènera moins de violations de SLA. Cela pose par conséquent la question sur la validité de la métrique qu'est la violation de SLA. En effet, si celle-ci est intéressante pour modéliser une certaine qualité de service, sa nature binaire empêche au niveau de la performance des services d'avoir un regard concret sur l'effet de telle ou telle allocation. Par exemple une tâche qui s'exécute à 50% de ce qu'elle avait demandé ne tournera pas bien, alors qu'une qui s'exécute à 99% du SLA aura seulement un léger ralentissement. Pourtant les deux compteront comme une violation de SLA.

D'un point de vue plus pratique au niveau du système de gestion des tâches dans le cloud, il apparaît important que le réallocateur de tâches doive avoir conscience des tendances globales d'utilisation des ressources du cluster. La raison en est simple, il sera important de faire un compromis entre la consolidation d'un côté, et l'équilibrage de l'autre. Si le cluster est en train

de se charger, il sera potentiellement plus judicieux de faire une faible consolidation avec un équilibrage plus important, alors qu'à l'inverse, si le système est en train de se décharger, il sera plus important de faire de la consolidation.

De plus, en termes de communication entre le gestionnaire et l'allocateur, il est important d'avoir une allocation en deux phases. La première qui tente de consolider selon la manière présentée, et la seconde qui équilibre la charge sur les hôtes conservés, c'est-à-dire ceux qui n'ont pas été choisi pour s'éteindre au prochain timestep. Cela pour éviter autant que possible d'avoir des hôtes tampons qui seront allumés sans utilisation de leur capacité de calcul.

Il est aussi intéressant d'étudier la possibilité de "fixer" une tâche sur un hôte. Que ce soit parce que celle-ci a une forte variabilité ou tout simplement parce qu'elle est volumineuse, migrer ce genre de tâches à problème nous enlève la capacité de prédire avec une précision suffisante l'état prochain du système, et amène des mauvaises allocations.

Enfin, il est de la plus haute importance de contrôler à quel point on veut consolider le système. Par exemple, une solution simple et pratique serait de limiter le nombre de machines virtuelles que l'on peut avoir sur un même hôte, et ce, en fonction du nombre de coeurs de cet hôte. Cela aurait pour effet de réduire les interactions néfastes entre les machines virtuelles qui s'exécutent, et limiterait aussi le risque de violation. Il y a moins de risques d'avoir des violations de SLA sur un hôte avec 2 VMs que sur un hôte avec 15 VMs. De plus, trop de consolidation entraine un nombre accru de violations pendant la durée des migrations. Cela représente un point d'inflexion dans l'économie d'énergie par rapport à la qualité de service. Par conséquent, le surcoût en énergie d'une consolidation plus faible vaudra largement la réduction en nombre de violations qui y sera associée. Trouver ce point d'inflexion peut se révéler compliqué, mais d'une manière générale, il sera préférable de moins charger les machines par rapport à une allocation statique.

Cependant, ces travaux reposent sur un nombre d'hypothèse dont il est bon de discuter. La première est celle du modèle de migration, ou plus généralement de l'intégration de la composante dynamique dans le modèle. On peut ainsi raisonnablement se poser la question de la validité du modèle de migration. Est-il cohérent d'avoir la même portion de ressource à la fois sur l'hôte source et sur l'hôte destination ? La manière dont est implémentée la migration dans Xen par exemple semble suggérer que ce ne soit pas tout à fait le cas, mais cette même implémentation suggère bien une surconsommation. Le modèle tel qu'implémenté dans nos travaux prend donc un cas conservatif, puisqu'on va à priori surévaluer dans la majorité des cas la consommation générée par la migration.

De plus, le temps induit par la migration n'est pas borné, c'est-à-dire que nous ne pouvons prédire combien de temps une certaine machine va migrer. Cette migration va prendre d'autant plus de temps qu'un goulot d'étranglement va se créer au niveau de l'infrastructure réseau. Ainsi, bien que notre infrastructure nous permette de ne pas définir une borne de temps précise pour les valeurs des timesteps, on pourra toujours borner ce temps, si tant est que nous intégrons dans notre infrastructure de cloud et dans nos algorithmes un nombre maximum de migrations. Cependant, une telle limitation n'est pas forcément nécessaire. Un bon algorithme de réallocation tentera de migrer le moins possible, ou plutôt tentera de fixer certaines tâches, à priori les plus lourdes, sur les hôtes afin de limiter autant que possible les migrations coûteuses, et ainsi limiter l'effet de goulot d'étranglement mentionné précédemment.

L'hétérogénéité peut aussi potentiellement amener des problèmes. Au niveau inter-cluster d'une part, puisque les liens réseaux inter-clusters sont moins performants que intra-cluster, mais aussi parce qu'on ne pourra pas utiliser l'image de la machine virtuelle sur un NAS dans un cluster différent de celui vers lequel la machine virtuelle va migrer. Ensuite, on peut se poser la question de l'effet que peut avoir l'hétérogénéité sur les résultats. En effet, peut-être qu'avoir un hôte ayant des plus grandes capacités que d'autes peut potentiellement réduire le coût relatif

de la migration vers et depuis cet hôte. Bien que cette hétérogénéité n'ait pas été testée dans nos expériences, notre infrastructure de tests a été faite de manière à gérer ce genre de cas. Par contre, nous avons choisi de ne pas l'intégrer aux expériences pour simplifier l'extraction de résultats, qui sont déjà suffisamment interdépendants.

Ouverture

On peut aussi discuter de comment améliorer les algorithmes. L'algorithme VectorPacking, comme nous avons pu le voir dans les expériences, tend à consolider de manière trop forte les tâches. Ainsi, une amélioration pourrait être de paramétrer l'algorithme afin de le rendre moins aggressif en terme de consolidation, plus souple. Ainsi, nous aurions un système moins contraint, et ainsi moins de violations de SLA dûs au manque de ressources. Une autre amélioration des algorithme consisterait à pouvoir prédire le nombre d'hôtes à éteindre. De la même manière, il serait intéressant aussi d'intégrer la stratégie d'extinction des hôtes dans l'algorithme de réallocation. Intégrer de telles considérations permettrait de pouvoir prédire quels hôtes vont rester allumés sur les t prochains timesteps, et ainsi potentiellement les utiliser. De ce fait, on pourrait réduire le taux de violation de SLA, sans pour autant impacter la consommation d'énergie.

Il serait intéressant d'apporter l'hétérogénéité et le refroidissement au modèle. Au niveau du refroidissement, on peut facilement imaginer qu'allumer et éteindre un CRAC/CRAH prenne du temps, et qu'il faille aussi intégrer ce temps au modèle. Cela amènerait nous amènerait par ailleurs à devoir prendre en compte les différences entre les différents temps (allumage d'un hôte, migration d'une machine virtuelle, etc.). Nous avons ainsi choisi de prendre un temps identique entre l'allumage et l'extinction des hôtes et de la migration des machines virtuelles. Cependant, cette hypothèse, bien que pratique, n'est pas fidèle au monde réel, et il faudra donc étudier l'effet des différents temps que prennent les différentes actions possibles sur l'infrastructure. Cette étude et prise en compte nous permettrai d'aller plus loin avec la prise en compte notamment de l'hétérogénéité sur plusieurs clusters, comme présentée dans le chapitre 4. Le gros problème de la prise en compte d'un cloud composé de plusieurs clusters, est que la migration pose un gros problème inter cluster. En effet, nous n'avons pas de stockage centralisé entre les clusters, et il faut migrer non seulement la mémoire de travail mais aussi l'image en système en entier ! Il faudra donc prendre en compte la migration inter cluster, qui est possible, en changeant le temps de migration en fonction du temps où l'on va, ainsi que favoriser les migrations intra cluster, qui seront bien plus rapides que les migrations inter cluster.

Enfin, le nombre d'hôte qu'il manque dans le cas d'un système sur-utilisé peut-être intéressant à calculer. Cela permettra une meilleure correspondance entre la partie gestion de machines physique et la partie réallocation, et surtout cela permettra d'être plus proactif et précis dans l'allumage des hôtes. Cependant, il est important de noter que calculer le nombre exact de machines à allumer est difficile puisque dépendant d'une l'allocation, et il faudra surtout faire attention à ne pas sur-provisionner le nombre de machines à allumer en étant trop proactifs.

Chapitre 6

Conclusion

6.1 Conclusion

Le cloud computing est en pleine expansion et tend à s'imposer comme un des paradigmes dominants dans le paysage informatique. Les infrastructures proposant des services de cloud computing deviennent donc de plus en plus nombreuses, et de plus en plus grosses pour répondre à cette demande croissante de services décentralisés. Cette augmentation amène bien évidemment divers problèmes, dont celui de la consommation d'énergie, ou celui de l'utilisation efficace des ressources. Il faut donc concevoir des techniques et outils afin de répondre à ces nouveaux besoins de gestion.

Dans cette thèse, nous avons tenté de résoudre certains aspects du problème en nous concentrant sur la gestion des tâches, plus particulièrement sur l'allocation et la réallocation des tâches. Nous avons introduit dans le chapitre 4 un modèle linéaire d'allocation de tâches statique basé sur un ensemble de contraintes linéaires. Ce modèle est basé sur les leviers proposés par la virtualisation et la migration des machines virtuelles et possède l'avantage de pouvoir d'une manière simple et relativement directe décrire l'existant, tout en étant facilement extensible et généralisable à d'autres ressources.

Le modèle ainsi défini nous a permis d'étudier les différents phénomènes impliqué dans les allocations, ainsi que les différentes interactions entre les paramètres. Nous avons donc pu définir un modèle multi objectifs, avec des objectifs antagonistes, que nous avons ensuite séparé en trois problèmes distincts : optimiser la performance à puissance électrique fixe, optimiser la puissance sous un minimum de performance à atteindre, et enfin optimiser les deux objectifs de manière concurrente avec une combinaison linéaire des deux. Ce faisant, nous avons défini différents algorithmes basés sur des heuristiques nous permettant de calculer des solutions proches de l'optimal en des temps raisonnables.

Nous avons ensuite étendu notre modèle pour y intégrer la prise en compte de l'hétérogénéité d'une part, et des infrastructures de refroidissement d'autre part. L'intégration de l'aspect hétérogénéité nous a permis d'être capable de passer d'un contexte de cloud monocluster à un contexte multicluster, et ainsi d'être plus proche de la réalité. L'aspect infrastructure de refroidissement est importante quand à elle puisque les infrastructure de refroidissement sont grosses consommatrices d'énergie, et participent ainsi de manière importante aux coûts opérationnels des centres de serveurs. Nous avons pu voir avec les expériences que nous avons conduites que ces différents aspects agissent sur la qualité des solutions, et que prendre en compte en amont des considérations d'hétérogénéité et de refroidissement amène de meilleurs résultats.

Nous avons pu voir qu'ajouter la prise en compte de l'hétérogénéité amenait une augmentation

du yield minimum allant jusqu'à 7% pour le problème BoundedPower par rapport à la non prise en compte de cette hétérogénéité. De la même manière pour le problème BoundedYield, nous avons pu observer une diminution de la consommation d'énergie de 4%.

Nous avons pu aussi voir lors de l'introduction des équipements de refroidissement, que prendre en compte dans l'algorithme de placement dans le problème BoundedYield amène une amélioration de la consommation d'énergie allant jusqu'à 11%. Pour le problème BoundedPower par contre, une amélioration du minimum yield de 2% à pu être observée par rapport à la non prise en compte du refroidissement.

Enfin, nous avons observé l'impact de ces deux aspects sur les résultats, en prenant en compte un seul des aspects, les deux, ou aucun. Nous avons pu ainsi observer un écart allant jusqu'à 28% en consommation d'énergie entre le moins bon (VP_CMAX_OVER_COOL, qui prend en compte uniquement le refroidissement) et le meilleur algorithme (VP_H_POWER, qui prend en compte l'hétérogénéité et le refroidissement). Ces deux algorithmes montrent aussi les meilleures et pires performances pour le problème BoundedPower, affichant un écart de performance de 16% de yield minimum entre le meilleur (VP_CMAX_OVER_COOL) et le pire (VP_H_POWER).

Dans le chapitre 5, nous avons ensuite étendu le modèle d'allocation statique, pour y introduire des considérations dynamique. Nous avons notamment introduit des considérations de temps de migration, de temps d'allumage et de temps d'extinction des hôtes, ainsi que les consommations de ressources qui y sont associées. La migration live des machines virtuelles et l'allumage et l'extinction des hôtes sont les pierres angulaires de la majorité des approches de gestion de tâches dans les clouds, et les coûts induits par la migration par exemple, sont importants et ne peuvent pas être ignorés longtemps. Ces considérations sont pourtant souvent ignorées ou trop simplifiées dans les travaux similaires [87, 101].

Nous sommes donc passé du statique au dynamique en étendant l'ensemble de contraintes à de nouvelles contraintes, et en étendant les variables pour y ajouter une nouvelle dimension, notant au passage l'explosion du nombre de variables et de contraintes même pour des petites instances. Par exemple, résoudre un système linéaire avec 3 tâches, 2 hôtes sur 4 timesteps reviendra à résoudre un système avec plus de 300 contraintes et 150 variables. Nous avons ensuite implémenté une infrastructure de simulation de cloud afin de tester la gestion de ces tâches dynamique via un gestionnaire autonome.

Comme nous avons pu le voir en section 5.3, l'infrastructure multi niveaux gérée de manière automatique a permi de réduire la consommation d'énergie allant de 11% jusqu'à 37% dans le meilleur des cas en ne prenant en compte que les algorithmes qui passent à l'échelle. Ces économies d'énergies ont pu être effectuées en gardant un taux de violations de SLA bas, allant de 0% pour des workloads stables à 8% dans le cas de workloads avec une grande volatilité.

6.2 Discussion et Perspectives

Durant les recherches que nous avons mené au cours de cette thèse, nous avons émis nombre d'hypothèses, soulevé des points nécessitant discussion et ouvert bon nombre de directions de recherche. Ces points se déclinent en trois catégories : d'abord l'adaptation possible du modèle à des infrastructures ou situations, ensuite l'extension du modèle à de nouveaux environnements, et enfin la validation du modèle actuel et la relaxation de certaines hypothèses.

6.2.1 Adaptabilité du modèle

Le premier point concerne l'adaptation du modèle défini dans les chapitres précédents. Ce modèle pose la question de savoir pourquoi nous n'avons pas utilisé les techniques de DVFS,

qui est une des techniques les plus répandues lorsque l'on s'intéresse aux économies d'énergies. Nous avons mentionné précédemment que les changements de fréquences avaient parfois un effet antagoniste avec les effets de l'allocation d'un pourcentage de ressources. En effet, pour être efficace, le DVFS doit être à l'échelle de l'application, c'est-à-dire doit réagir aux différents changements de celle-ci, et cela peut aller à l'encontre d'une allocation, qui alloue comme dans notre cas des ressources selon une certaine fréquence sur une période donnée. De plus, même si l'on ne prend pas en compte les effets négatifs que peut avoir le DVFS sur certaines parties d'une machine, nous pouvons l'utiliser. Il faut aussi souligner qu'il existe des couples voltage/fréquence optimaux en terme de performance et d'énergie qui pourrait nous permettre d'avoir les hôtes toujours réglés sur cette fréquence.

Alors comment intégrer le DVFS à nos travaux ? On peut tout d'abord supposer que toutes les machines ont été réglées à leur couple optimal, et itérer à partir de là. Mais cette solution, bien que valide dans une certaine mesure, n'est que peu satisfaisante, puisqu'elle présuppose une connaissance des applications sous-jacentes ainsi qu'une connaissance de l'infrastructure.

Une piste serait donc d'intégrer le DVFS à l'intérieur du modèle, en prenant les différents états de la fréquence d'un processeur comme des hôtes différents. Il faudrait alors rajouter une contrainte d'exclusivité entre les différents hôtes, ou plutôt entre les différentes fréquences d'un même hôte. Nous aurions ainsi une allocation qui serait en théorie effectuée sur un hôte s'exécutant à sa fréquence optimale. Cette approche est intéressante à utiliser puisqu'elle amènera encore plus d'économies d'énergies sans pour autant pénaliser la performance, elle sera pourtant au coût d'un temps de calcul significativement plus important, du fait de la recherche supplémentaire du couple adéquat. Pourtant, utiliser de la sorte le DVFS nous empêchera de l'utiliser à un niveau plus bas, celui de l'application elle même, et amènera probablement moins d'économies d'énergies qu'une telle approche. La meilleure utilisation possible du DVFS se fera au niveau de l'application, puisqu'on pourrait potentiellement réagir à chaque changement de charge de l'application en attribuant le couple fréquence voltage approprié. Utiliser le DVFS à un niveau plus haut, et par conséquent sur un temps plus grand puisque c'est le gestionnaire qui va définir le couple pour la durée de l'allocation, amènera donc un couple sous optimal sur la durée de l'allocation, et donc une consommation d'énergie plus grande. Restera à déterminer si avoir un gestionnaire de ressource indépendant d'un scheduler processeur avec DVFS est plus ou moins intéressant en terme d'énergie et de performance qu'avoir une approche complètement intégrée.

Nous pourrions aussi rajouter la gestion de la fréquence au niveau du gestionnnaire de cloud tel que présenté dans le chapitre 5, au niveau de la reconfiguration des machines virtuelles. Par exemple, nous pourrions avoir des règles de la même manière que pour la gestion de la reconfiguration de machines virtuelles effectuant des réductions ou augmentation de la fréquence du processeur en fonction du nombre de tâches sur la machine et de la charge de cette machine. Il faudra cependant prendre garde à utiliser de telle technique de manière conservative, c'est-à-dire modérément aggressive sur les décisions de changement de fréquence, et il faudra décider de la précédence entre la reconfiguration et le changement de fréquence, afin de savoir si le système va d'abord chercher à baisser la fréquence ou à augmenter l'allocation, ou les deux à la fois. Cette technique simple pourrait nous permettre de réduire la consommation de puissance des allocations dans certains états (choisis par expérimentations préalables) des machines.

Enfin, nous pourrions regarder du coté du *power clamping/capping* [94]. Ces nouvelles techniques introduites notamment dans les familles de processeur Sandy Bridge d'Intel, permettent de définir des bornes de puissance à être consommée par le processeur, celui-ci s'assurant matérielle-ment que cette borne soit respectée. Ce type de technique serait amené à remplacer ou à être en complément du DVFS. Cependant, la grande majorité des processeurs dans les serveurs actuels ne possèdent pas ce genre de technique, et il faudra un certain temps pour avoir une diffusion de cette technique. Nous pouvons par contre remarquer que le power capping similaire à notre

modèle dans le cas du problème BoundedPower, et que nous pourrions l'y adapter à priori de manière efficace.

Le modèle que nous avons défini est aussi facilement adaptable et de manière directe à certaines des nouvelles architectures dont l'état d'allumage des processeurs (ou coeurs) [74] peut être contrôlé pendant l'exécution d'une application. Ainsi, si l'on considère une machine physique comme un microcluster, le modèle s'applique de manière directe. On pourrait ainsi consolider les machines virtuelles sur un sous-ensemble réduit de coeurs pour éteindre les autres, si l'on pose comme hypothèse que nous séparons la mémoire de manière exclusive entre les coeurs. Mais cela est le moindre des problèmes. Il faudra modifier le modèle de la consommation d'énergie, car si l'on éteint un coeur alors que les autres sont toujours allumés, la machine physique est toujours allumée et la consommation d'énergie n'est pas réduite à 0. Pour palier à ce problème, on peut penser à une infrastructure de décision hiérarchique comprenant une composante décisionnelle sur chaque machine physique, qui fera une allocation energy-aware selon le même modèle, mais à l'échelle des processeurs. Ainsi, une premiere allocation pour décider quelles tâches vont sur quelles machines serait effectuée, en prenant la capacité en ressources d'une machine physique comme la somme des capacités de ses coeurs, et une seconde réarrangerait les tâches sur la machine pour tenter d'en éteindre des coeurs. Cependant, ce genre d'approche demande de se poser d'autant plus la question de l'impact qu'ont des machines virtuelles sur un même coeur, puisque l'on sait déjà que des machines virtuelles allouées sur la même machine physique s'impactent entre elles [81].

6.2.2 Extension du modèle

Outre l'adaptation de notre modèle avec des changements mineurs pour se conformer à des paradigmes et infrastructures différents, il est possible d'étendre le modèle pour prendre en compte de nouveaux aspects, et relâcher certaines hypothèses.

Par exemple, de la même manière que pour les coeurs des processeurs qu'il serait possible d'éteindre indépendament les uns des autres, on peut généraliser cet aspect à l'extinction de toute partie, si tant est que cela est possible, d'une machine physique. Pour la mémoire, celle-ci est considérée ici comme une ressource rigide, et par conséquent l'ajout d'un coût en fonction de combien de mémoire est consommée est relativement direct. Ainsi, le compromis qui doit être fait avec la perte de performance à lieu à une échelle bien plus petite que celle de l'allumage d'un hôte.

Une telle généralisation par contre, mettrait l'accent sur une proportionnalité énergétique plus grande, et les algorithmes proposés ici pourraient avoir des résultats moindre. Ainsi, les algorithmes que nous avons implémentés ont des performances basées sur l'hypothèse que la majorité de la consommation d'énergie d'une machine vient du fait d'être allumée. Cependant, si l'on suppose que toute partie peut être éteinte, cette différence faiblit grandement, et la consolidation, qui est la force par exemple de l'algorithme VectorPack n'aurait pas autant d'importance, ce qui amènerait des performances moindre. La majorité des économies d'énergies viendrait dans ce cas de la reconfiguration des machines virtuelles, et il faudrait prendre garde à ne pas tenter de faire une trop grande consolidation, afin de ne pas nuire à la reconfiguration. En effet, une bonne reconfiguration nécessite de la marge de manoeuvre, idéalement une seule tâche sur un hôte.

Cette généralisation de l'état peut être à une échelle plus petite en relachant l'hypothèse de l'état binaire allumé/éteint des hôtes. Il est possible de mettre les machines dans un état d'hibernation, qui se situe entre l'état allumé et éteint. Le passage de l'état hibernation à l'état allumé est bien plus rapide que le passage d'éteint à allumé. Les contraintes de non exécution de tâches sur un hôte en hibernation restent les mêmes, mais le réallumage est plus rapide et la machine consomme de l'électricité. On peut étendre ceci à plus d'état plus ou moins profonds

de sommeil, mais le principe reste le même. Il faut prendre en compte cet effet et étendre le modèle à ces nouveaux états. On peut donc définir, un nombre de timesteps pour l'allumage et l'extinction des hôtes, ainsi que pour tous les états intermédiaires. On peut donc imaginer qu'il faille 1 timestep pour passer de l'état d'hibernation à l'état allumé, alors qu'il en faut 3 depuis l'extinction totale. Cela pourra se faire avec une modification des contraintes linéaires au niveau du modèle, pour intégrer les états d'une part, et la durée en tant que paramètre d'autre part.

Rajouter des états pour l'infrastructure peut se faire aussi au niveau du cooling, qui consommera plus ou moins d'énergie en fonction de l'utilisation globale des hôtes associés. Du fait de la nature quadratique de la consommation du refroidissement, on peut facilement penser qu'étendre le coût de l'infrastructure de refroidissement à un coût linéaire comme celui du processeur peut ne pas être suffisant. On pourrait donc imaginer passer à un modèle de coût linéaire par morceau, afin de rester toujours en contrainte linéaire, et ainsi passer à un modèle plus proche de la réalité. Par exemple on peut imaginer qu'un CRAC/CRAH consomme $P = utilisation \times a$ watts en dessous de 50% de charge, et $P = utilisation \times 2a$ watts au dessus. Cette extension simple nous permet une amélioration majeure, qui est de prévenir les points chauds dans le cluster.

Comme mentionné dans la partie 5.3.1, il faut faire le parallèle entre le yield et les SLA. Nous avons choisi pour notre modèle d'utiliser le yield minimum pour des raisons d'équité. Nous avons ensuite adapté le modèle pour y intégrer des considérations de SLA. Il faudra cependant améliorer le modèle pour pouvoir passer à la fois aux considérations de minimisation des violations de SLA et d'équité. Le modèle actuel de violations de SLA est binaire, soit il y a une violation de SLA, soit non. On peut ainsi minimiser les violations tout en n'étant pas équitable, dans le cas d'une allocation où on alloue toutes les tâches sans violation de SLA, sauf une seule qui est proche de 0% de ce qu'elle avait requis, on aura alors un nombre minimum de violations mais une allocation qui ne peut pas être considérée comme bonne. Il faut ainsi intégrer une marge de manoeuvre dans les violations de SLA. Seulement la manière dont nous avons intégré les violations de SLA ne suffit pas, puisqu'il faudrait intégrer à quel point un SLA est dépassé, et pas seulement si il est dépassé, puisque 1% d'allocation en dessous de ce qui était requis n'aura pas le même effet sur la qualité d'expérience de l'utilisateur que d'allouer a 50% en dessous.

6.2.3 Etudes supplémentaires

Enfin, le dernier point de recherches supplémentaires concerne les expériences que nous avons faites, et les différentes études que nous pouvons faire.

Quel objectif pour les économies d'énergies ?

La première direction dans ce contexte est celle de l'étude de la différence entre puissance et énergie. En effet, si l'on assimilait la puissance à l'énergie dans le chapitre 4, du fait de l'hypothèse des tâches infinies, lorsque nous avons supprimé cette hypothèse pour passer à une réallocation dynamique, nous somme passés dans des considérations d'énergie, où l'énergie est la somme des puissances consommées à chaque timesteps. Ainsi, dans le cadre du problème borné en puissance, on peut se poser la question de savoir si il est plus intéressant de raisonner en terme d'énergie ou de puissance, à savoir s'il est plus intéressant d'optimiser l'énergie sur chaque timesteps, ou plutôt d'optimiser l'énergie globale.

Si l'on prend le problème de puissance pour des considérations matérielles de consommation d'électricité, à savoir on veut éviter que l'infrastructure dépasse une consommation donnée, la question ne se pose pas puisqu'il faut raisonner par timesteps afin de ne jamais dépasser cette limite. Par contre, si le but est de ne pas dépasser une certaine puissance pour budgétiser la consommation, ou tout simplement de réduire la consommation d'énergie, la question se pose.

Par conséquent, bien que les deux solutions soient viables, une étude sur ces phénomènes peut nous amener des éléments de réponse pour modifier l'infrastructure de cloud pour prendre en compte ces deux aspects. Il est probable que raisonner de manière globale sur la durée de l'exécution donnera de meilleurs résultats, mais ce sera si et seulement si on a une connaissance parfaite des demandes des tâches. Ainsi, si les résultats seront meilleurs, cela sera probablement au prix d'une plus grande variabilité dans les résultats de ce fait, alors qu'optimiser par timestep donnera des résultats à priori plus stable. C'est en fait tout le problème de la prédiction. Si la prédiction est suffisamment fiable, on préferera raisonner en connaissance du futur, alors que si la charge de travail est trop variable, on préfèrera raisonner sur le présent. Dans tous les cas, il apparait nécessaire d'implémenter ces deux aspects (à la fois dans les algorithmes et les seuils) afin de raisonner dans le meilleur environnement possible.

Influence des différents temps d'action

Ensuite, l'étude du temps de migration sera intéressante. Le temps de migration est au coeur des problèmes liés à la réallocation des machines virtuelles. Tester l'influence du temps de migration, avec l'influence du comportement de la machine virtuelle et de l'implémentation de la migration, est nécessaire pour adapter les algorithmes, afin de pouvoir modifier les algorithmes de décision de manière adéquate. Il sera intéressant de voir à quel point un temps long de migration rend les algorithmes moins bons, et comment ne pas migrer certaines machines sous certaines contraintes (si elle est trop demandeuse de RAM par exemple) améliorera les résultats dans ces cas là.

Etude du surprovisionnement

Les machines virtuelles doivent aussi faire l'objet d'une étude plus minutieuse. Nous avons supposé qu'il n'y avait pas d'erreur dans les mesures de consommation, et que le client connaissait ses besoins. Nous avons ensuite un peu relâché cette hypothèse en réduisant l'allocation en dessous de ce que la tâche avait requis pour être au plus près de ce qu'elle avait consommé. Cependant, il est intéressant de voir comment ces différences influencent le système et les résultats, comment elles peuvent être rattrapées par le système. Il faut un algorithme de décision capable d'absorber cette marge d'erreur via notamment les seuils définis en partie 5.3.1.

De la même manière, il est intéressant d'étudier le surprovisionnement que l'on peut faire pendant l'allocation de tâche. Par exemple, lorsque l'on tente d'allouer un ensemble de tâches, et que l'allocation résultante fait dépasser d'1% une seule tâche, l'allocation échoue. Pourtant, il peut être intéressant de garder cette allocation en dépit du fait qu'elle soit considérée mauvaise, en rajoutant le taux d'erreur possible.

Expérimentations en environnement réel

Enfin, il faudra effectuer des expérimentations en situation réelle. Bien que non présentes dans ce manuscrit, nous allons effectuer des expérimentations implémentant certains scénarios décrits dans les chapitres précédents en situation réelle à l'aide de gestionnaire de clouds tels que Snooze[44], qui est très compatible au niveau fonctionnement avec notre approche, et ainsi avoir une validation expérimentale des résultats présentés précédemment, notamment vis-à-vis des temps de migration et d'allumage/extinction des hôtes, et du fonctionnement en timesteps en général. Pour ce faire, il faudra s'abstraire du fonctionnement par surcharge/souscharge de Snooze pour en faire une gestion par période de temps.

Table des figures

Liste des tableaux

Bibliographie

[1] Ilog cplex. http://www.ilog.com/products/cplex/.

[2] SPECweb99 Benchmark. http://www.spec.org/web99/, 1999-2000.

[3] Grid'5000 : A Large Scale And Highly Reconfigurable Experimental Grid Testbed. *International Journal of High Performance Computing Applications*, 20(4) :481–494, November 2006.

[4] COST IC0804 - Energy efficiency in large scale distributed systems. http://www.cost804.org, 2009-2013.

[5] ASHRAE Technical Committee (TC) 9.9. 2011 Thermal Guidelines for Data Processing Environments. http://www.eni.com/green-data-center/it_IT/static/pdf/ASHRAE_1.pdf, 2011.

[6] Yuki Abe, Hiroshi Sasaki, Martin Peres, Koji Inoue, Kazuaki Murakami, and Shinpei Kato. Power and performance analysis of gpu-accelerated systems. In *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, HotPower'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[7] International Atomic Energy Agency. Power Reactor Information System. http://pris.iaea.org/PRIS/CountryStatistics/CountryDetails.aspx?current=FR, Jan 2013.

[8] United States Environmental Protection Agency. Report to congress on server and data center energy efficiency. http://www.energystar.gov/index.cfm?c=prod_development.server_efficiency_study, 2007.

[9] S. Akoush, R. Sohan, A. Rice, A.W. Moore, and A. Hopper. Predicting the performance of virtual machine migration. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 37–46, 2010.

[10] Samer Al-Kiswany, Dinesh Subhraveti, Prasenjit Sarkar, and Matei Ripeanu. Vmflock : virtual machine co-migration for the cloud. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 159–170, New York, NY, USA, 2011. ACM.

[11] Amazon. Amazon CloudFront Presentation. http://aws.amazon.com/fr/cloudfront/, Sept 2012.

[12] Vlasia Anagnostopoulou, Heba Saadeldeen, and Frederic T. Chong. Quantifying the environmental advantages of large-scale computing. In *Proceedings of the International Conference on Green Computing*, GREENCOMP '10, pages 269–280, Washington, DC, USA, 2010. IEEE Computer Society.

[13] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks, summary and preliminary results. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, Supercomputing '91, pages 158–165, New York, NY, USA, 1991. ACM.

[14] Bharathan Balaji, John McCullough, Rajesh K. Gupta, and Yuvraj Agarwal. Accurate characterization of the variability in power consumption in modern mobile processors. In *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, HotPower'12, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.

[15] Ayan Banerjee, Tridib Mukherjee, Georgios Varsamopoulos, and Sandeep K. S. Gupta. Cooling-aware and thermal-aware workload placement for green hpc data centers. In *Proceedings of the International Conference on Green Computing*, GREENCOMP '10, pages 245–256, Washington, DC, USA, 2010. IEEE Computer Society.

[16] Prithviraj Banerjee, Chandrakant Patel, Cullen Bash, Amip Shah, and Martin Arlitt. Towards a net-zero data center. *J. Emerg. Technol. Comput. Syst.*, 8(4) :27 :1–27 :39, November 2012.

[17] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[18] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5) :164–177, October 2003.

[19] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12) :33–37, December 2007.

[20] Geoffrey C. Bell and Cliff Federspiel. Demonstration of Datacenter Automation Software and Hardware (DASH) at the California Franchise Tax Board California Energy Commission. . `http://gaia.lbl.gov/btech/papers/3131.pdf`, december 2009.

[21] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.*, 28(5) :755–768, May 2012.

[22] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 215–224, New York, NY, USA, 2010. ACM.

[23] Damien Borgetto, Henri Casanova, Georges Da Costa, and Jean-Marc Pierson. Energy-Aware Service Allocation. *Future Generation Computer Systems*, 28(5) :769–779, mai 2012. Special section on Energy Efficiency in Large-Scale Distributed Systems.

[24] Damien Borgetto, Georges Da Costa, Jean-Marc Pierson, and Amal Sayah. Energy-Aware Resource Allocation (regular paper). In *Energy Efficient Grids, Clouds and Clusters Workshop (co-located with Grid) (E2GC2), Banff, 13/10/2009-15/10/2009*, page (electronic medium), http ://www.ieee.org/, octobre 2009. IEEE.

[25] Damien Borgetto, Michael Maurer, Georges Da Costa, Ivona Brandic, and Jean-Marc Pierson. Energy-efficient and SLA-Aware Management of IaaS Clouds (regular paper). In *ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy), Madrid, 09/05/2012-11/05/2012*, page (electronic medium), http ://portal.acm.org/dl.cfm, 2012. ACM DL.

[26] David Breitgand, Gilad Kutiel, and Danny Raz. Cost-aware live migration of services in the cloud. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, SYSTOR '10, pages 11 :1–11 :1, New York, NY, USA, 2010. ACM.

[27] Tom Brey (IBM), Pamela Lembke (IBM), Joe Prisco (IBM), Ken Abbott (Emerson), Dominic Cortese (Emerson), Kerry Hazelrigg (Disney), Jim Larson (Disney), Stan Shaffer (MSI), Travis North (Charsworth Products), and Tommy Darby (Texas Instruiments). EWP#39-Case Study : The ROI of Cooling System Energy Efficiency Upgrades. `http://www.thegreengrid.org/en/Global/Content/white-papers/CaseStudyROIofCoolingSystemEnergyEfficiencyUpgrades`, June 2011.

[28] Thomas D. Burd and Robert W. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of the 2000 international symposium on Low power electronics and design*, ISLPED '00, pages 9–14, New York, NY, USA, 2000. ACM.

[29] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. *SIGOPS Oper. Syst. Rev.*, 35(5) :103–116, October 2001.

[30] Fabio Checconi, Tommaso Cucinotta, and Manuel Stein. Real-time issues in live migration of virtual machines. In *Proceedings of the 2009 international conference on Parallel processing*, Euro-Par'09, pages 454–466, Berlin, Heidelberg, 2010. Springer-Verlag.

[31] Yuan Chen, Daniel Gmach, Chris Hyser, Zhikui Wang, Cullen Bash, Christopher Hoover, and Sharad Singhal. Integrated management of application performance, power and cooling in data centers. In *NOMS*, pages 615–622. IEEE, 2010.

[32] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[33] Intel Corporation. Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor White Paper . `ftp://download.intel.com/design/network/papers/30117401.pdf`, March 2004.

[34] Standard Performance Evaluation Corporation. SPEC Power benchmark results. `http://www.spec.org/power_ssj2008/results/res2012q1/`, March 2012.

[35] G. Da Costa, J.-P. Gelas, Y. Georgiou, L. Lefevre, A.-C. Orgerie, J. Pierson, O. Richard, and K. Sharma. The green-net framework : Energy efficiency in large scale distributed systems. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, 2009.

[36] Georges Da Costa, Marcos Dias de Assunção, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard, and Amal Sayah. Multi-facet approach to reduce energy consumption in clouds and grids : the green-net framework. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 95–104, New York, NY, USA, 2010. ACM.

[37] Ulrich Drepper. The cost of virtualization. *Queue*, 6(1) :28–35, January 2008.

[38] Corentin Dupont, Giovanni Giuliani, Fabien Hermenier, Thomas Schulze, and Andrey Somov. An energy aware framework for virtual machine placement in cloud federated data centres. In *Proceedings of the 3rd International Conference on Future Energy Systems*, e-Energy 2012, 2012.

[39] Truong Vinh Truong Duy, Y. Sato, and Y. Inoguchi. Performance evaluation of a Green Scheduling Algorithm for energy savings in Cloud computing. In *Parallel &amp ; Dis-*

tributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, pages 1–8. IEEE, April 2010.

[40] Dmytro Dyachuk and Michele Mazzucco. On allocation policies for power and performance. *CoRR*, abs/1102.3272, 2011.

[41] Vincent C. Emeakaroha, Pawel P. Labaj, Michael Maurer, Ivona Brandic, and David P. Kreil. Optimizing bioinformatics workflows for data analysis using cloud management techniques. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, WORKS '11, pages 37–46, New York, NY, USA, 2011. ACM.

[42] M. Etinski, J. Corbalan, J. Labarta, M. Valero, and A. Veidenbaum. Power-aware load balancing of large scale mpi applications. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, 2009.

[43] Maja Etinski, Julita Corbalan, Jesus Labarta, and Mateo Valero. Optimizing job performance under a given power constraint in hpc centers. In *Proceedings of the International Conference on Green Computing*, GREENCOMP '10, pages 257–267, Washington, DC, USA, 2010. IEEE Computer Society.

[44] Eugen Feller, Louis Rilling, Christine Morin, Renaud Lottiaux, and Daniel Leprince. Snooze : A Scalable, Fault-Tolerant and Distributed Consolidation Manager for Large-Scale Clusters. Rapport de recherche 7398, September 2010.

[45] Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Thermal-aware global real-time scheduling and analysis on multicore systems. *Journal of Systems Architecture - Embedded Systems Design*, 57(5) :547–560, 2011.

[46] A. Gandhi, Yuan Chen, D. Gmach, M. Arlitt, and M. Marwah. Minimizing data center sla violations and power consumption via hybrid resource provisioning. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, pages 1–8, Washington, DC, USA, 2011. IEEE Computer Society.

[47] R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman, 1979.

[48] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya. Energy-efficient scheduling of hpc applications in cloud computing environments. *CoRR*, abs/0909.1146, 2009.

[49] Rong Ge, Xizhou Feng, and Kirk W. Cameron. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 34–, Washington, DC, USA, 2005. IEEE Computer Society.

[50] GLPK. GNU Linear Programming Kit. `http://www.gnu.org/software/glpk`.

[51] Google. Google Data Centers. `https://maps.google.com/maps/ms?msid=200786673987469209255.0004b37b140b542230825`, Dec 2011.

[52] Google. The Big Picture. `http://www.google.com/green/bigpicture/#/intro/infographics-1`, Jan 2013.

[53] Chamara Gunaratne, Kenneth Christensen, Bruce Nordman, and Stephen Suen. Reducing the energy consumption of ethernet with adaptive link rate (alr). *IEEE Transactions on Computers*, 57 :448–461, 2008.

[54] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2012.

[55] László Gyarmati and Tuan Anh Trinh. How can architecture help to reduce energy consumption in data center networking? In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 183–186, New York, NY, USA, 2010. ACM.

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

[57] Junichi Hikita, Akio Hirano, and Hiroshi Nakashima. Saving 200kw and $200 k/year by power-aware job/machine scheduling. *Parallel and Distributed Processing Symposium, International*, 0 :1–8, 2008.

[58] Michael R. Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 51–60, New York, NY, USA, 2009. ACM.

[59] Marko Hoyer, Kiril Schröder, and Wolfgang Nebel. Statistical static capacity management in virtualized data centers supporting fine grained qos specification. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 51–60, New York, NY, USA, 2010. ACM.

[60] S. Huang and W. Feng. Energy-efficient cluster computing via accurate workload characterization. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 68–75, Washington, DC, USA, 2009. IEEE Computer Society.

[61] Sun Microsystems Inc. Sun^{TM} modular datacenter s20/d20 overview. `http://docs.oracle.com/cd/E19115-01/mod.dc.d20/820-5770-10/820-5770-10.pdf`, 2008.

[62] Greenpeace International. Make IT Green : Cloud Computing and its Contribution to Climate Change. `http://www.greenpeace.org/usa/en/mediacenter/reports/makeit-greencloudcomputing/`, 2010.

[63] Gihun Jung and Kwang Mong Sim, editors. *Location-Aware Dynamic Resource Allocation Model for Cloud Computing Environment*, 2012.

[64] Bithika Khargharia, Salim Hariri, and Mazin S. Yousif. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2) :167–181, June 2008.

[65] Avi Kivity, Uri Lublin, and Anthony Liguori. kvm : the linux virtual machine monitor. *Reading and Writing*, 1(1465-6914 (Electronic) LA - ENG PT - JOURNAL ARTICLE) :225–230, 2007.

[66] Data Center Knoweledge. Who owns the most servers? `http://www.datacenterknowledge.com/archives/2009/05/14/whosgotthemost-webservers/`, May 2009.

[67] Jonathan Koomey. Growth in data center electricity use 2005 to 2010. 2011.

[68] Jonathan G. Koomey, Stephen Berard, Marla Sanchez, and Henry Wong. Assessing trends in the electrical efficiency of computation over time.

[69] Krzysztof Kurowski, Ariel Oleksiak, Michal Witkowski, and Jarek Nabrzyski. Distributed power management and control system for sustainable computing environments. In *Proceedings of the International Conference on Green Computing*, GREENCOMP '10, pages 365–372, Washington, DC, USA, 2010. IEEE Computer Society.

[70] Eun Kyung Lee, Hariharasudhan Viswanathan, and Dario Pompili. Vmap : Proactive thermal-aware virtual machine allocation in hpc cloud datacenters.

[71] Young Choon Lee and Albert Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 92–99, Washington, DC, USA, 2009. IEEE Computer Society.

[72] Laurent Lefèvre and Anne-Cécile Orgerie. Designing and Evaluating an Energy Efficient Cloud. *Journal of Supercomputing*, 51(3) :352–373, March 2010.

[73] William Leinberger, George Karypis, and Vipin Kumar. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *ICPP*, pages 404–412, 1999.

[74] Jacob Leverich, Matteo Monchiero, Vanish Talwar, Parthasarathy Ranganathan, and Christos Kozyrakis. Power management of datacenter workloads using per-core power gating. *IEEE Comput. Archit. Lett.*, 8(2) :48–51, July 2009.

[75] Harold Lim, Aman Kansal, and Jie Liu. Power budgeting for virtualized data centers. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 5–5, Berkeley, CA, USA, 2011. USENIX Association.

[76] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 171–182, New York, NY, USA, 2011. ACM.

[77] Liang Liu, Hao Wang, Xue Liu, Xing Jin, Wen Bo He, Qing Bo Wang, and Ying Chen. Greencloud : a new architecture for green data center. In *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, ICAC-INDST '09, pages 29–38, New York, NY, USA, 2009. ACM.

[78] Anne-Cécile Orgerie Marcos Dias de Assunçao and Laurent Lefèvre. Ict energy logs. `http://www.ens-lyon.fr/LIP/RESO/ict-energy-logs/`, 2009-2010.

[79] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Simulating autonomic sla enactment in clouds using case based reasoning. In Elisabetta Nitto and Ramin Yahyapour, editors, *Towards a Service-Based Internet*, volume 6481 of *Lecture Notes in Computer Science*, pages 25–36. Springer Berlin Heidelberg, 2010.

[80] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Enacting slas in clouds using rules. In *Proceedings of the 17th international conference on Parallel processing - Volume Part I*, Euro-Par'11, pages 455–466, Berlin, Heidelberg, 2011. Springer-Verlag.

[81] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds : managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 237–250, New York, NY, USA, 2010. ACM.

[82] Ripal Nathuji, Ankit Somani, Karsten Schwan, and Yogendra Joshi. Coolit : coordinating facility and it management for efficient datacenters. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 15–15, Berkeley, CA, USA, 2008. USENIX Association.

[83] Dusit Niyato, Sivadon Chaisiri, and Lee Bu Sung. Optimal power management for server farm to support green computing. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 84–91, Washington, DC, USA, 2009. IEEE Computer Society.

[84] Oracle. Strategies for Solving the Datacenter Space, Power, and Cooling Crunch : Sun Server and Storage Optimization Techniques. `http://www.oracle.com/us/products/serversstorage/servers/sparc-enterprise/sundatacenterspacepowerwp075961.pdf`, March 2010.

[85] Anne-Cécile Orgerie, Laurent Lefèvre, and Jean-Patrick Gelas. Save watts in your grid : Green strategies for energy-aware framework in large scale distributed systems. In *Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*, ICPADS '08, pages 171–178, Washington, DC, USA, 2008. IEEE Computer Society.

[86] Steven Pelley, David Meisner, Thomas F Wenisch, and James W Vangilder. Understanding and abstracting total data center power. pages 2706–2710, 2009.

[87] Vinicius Petrucci, Enrique V. Carrera, Orlando Loques, Julius C. B. Leite, and Daniel Mosse. Optimized management of power and performance for virtualized heterogeneous server clusters. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 23–32, Washington, DC, USA, 2011. IEEE Computer Society.

[88] Jean-Marc Pierson and Henri Casanova. On the utility of dvfs for power-aware job placement in clusters. In *Proceedings of the 17th international conference on Parallel processing - Volume Part I*, Euro-Par'11, pages 255–266, Berlin, Heidelberg, 2011. Springer-Verlag.

[89] Meikel Poess and Raghunath Othayoth Nambiar. Energy cost, the key challenge of today's data centers : a power consumption analysis of tpc-c results. *Proc. VLDB Endow.*, 1(2) :1229–1240, August 2008.

[90] Google Research. Toward energy-proportionnal data centers. `http://googleresearch.blogspot.fr/2010/09/towardsenergyproportional-datacenters.html`, September 2010.

[91] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 3–3, Berkeley, CA, USA, 2008. USENIX Association.

[92] I. Rodero, J. Jaramillo, A. Quiroz, M. Parashar, F. Guim, and S. Poole. Energy-efficient application-aware online provisioning for virtualized clouds and data centers. In *Proceedings of the International Conference on Green Computing*, GREENCOMP '10, pages 31–45, Washington, DC, USA, 2010. IEEE Computer Society.

[93] Ivan Rodero, Hariharasudhan Viswanathan, EunKyung Lee, Marc Gamell, Dario Pompili, and Manish Parashar. Energy-efficient thermal-aware autonomic management of virtualized hpc cloud infrastructure. *Journal of Grid Computing*, 10(3) :447–473, 2012.

[94] Barry Rountree, Dong H. Ahn, Bronis R. de Supinski, David K. Lowenthal, and Martin Schulz. Beyond dvfs : A first look at performance under a hardware-enforced power bound. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, IPDPSW '12, pages 947–953, Washington, DC, USA, 2012. IEEE Computer Society.

[95] Jan Hendrik Schönherr, Jan Richling, Matthias Werner, and Gero Mühl. Event-driven processor power management. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 61–70, New York, NY, USA, 2010. ACM.

[96] Venkateswaran Shekar and Baback Izadi. Energy aware scheduling for dag structured applications on heterogeneous and dvs enabled processors. In *Proceedings of the International*

Conference on Green Computing, GREENCOMP '10, pages 495–502, Washington, DC, USA, 2010. IEEE Computer Society.

[97] Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. Server-storage virtualization : integration and load balancing in data centers. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 53 :1–53 :12, Piscataway, NJ, USA, 2008. IEEE Press.

[98] Top 500 Supercomputer sites. Top 500 supercomputer list of june 2012. `http://www.top500.org/lists/2012/06`, june 2012.

[99] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 10–10, Berkeley, CA, USA, 2008. USENIX Association.

[100] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova. Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and Distributed Computing*. in press (Digital Object Identifier : 10.1016/j.jpdc.2010.05.006).

[101] Mark Stillwell, David Schanzenbach, Frederic Vivien, and Henri Casanova. Resource allocation using virtual clusters. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 260–267, Washington, DC, USA, 2009. IEEE Computer Society.

[102] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds : towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1) :50–55, December 2008.

[103] H. Viswanathan, E. K. Lee, I. Rodero, D. Pompili, M. Parashar, and M. Gamell. Energy-aware application-centric vm allocation for hpc workloads. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, pages 890–897, Washington, DC, USA, 2011. IEEE Computer Society.

[104] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds : A performance evaluation. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 254–265, Berlin, Heidelberg, 2009. Springer-Verlag.

[105] Lizhe Wang, Gregor von Laszewski, Jay Dayal, and Fugang Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 368–377, Washington, DC, USA, 2010. IEEE Computer Society.

[106] Andrew J. Younge, Gregor von Laszewski, Lizhe Wang, Sonia Lopez-Alarcon, and Warren Carithers. Efficient resource management for cloud computing environments. In *Proceedings of the International Conference on Green Computing*, GREENCOMP '10, pages 357–364, Washington, DC, USA, 2010. IEEE Computer Society.

[107] Dakai Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, ICCAD '04, pages 35–40, Washington, DC, USA, 2004. IEEE Computer Society.