Évaluation et optimisation de performance énergétique des centres de calcul

Georges da Costa

THÈSE

En vue de l'obtention de l'

HABILITATION DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *12 novembre 2015* **par :**

GEORGES DA COSTA

Évaluation et optimisation de performance énergétique des centres de calcul

---

JURY

CHRISTINE MORIN	Directeur de recherche	Rapporteur
OLIVIER BEAUMONT	Directeur de recherche	Rapporteur
PIERRE SENS	Professeur d'Université	Rapporteur
FRÉDÉRIC DESPREZ	Directeur de recherche	Membre du Jury
LAURENT LEFÈVRE	Chargé de recherche, HDR	Membre du Jury
RAYMOND NAMYST	Professeur d'Université	Membre du Jury
JEAN-MARC PIERSON	Professeur d'Université	Membre du Jury

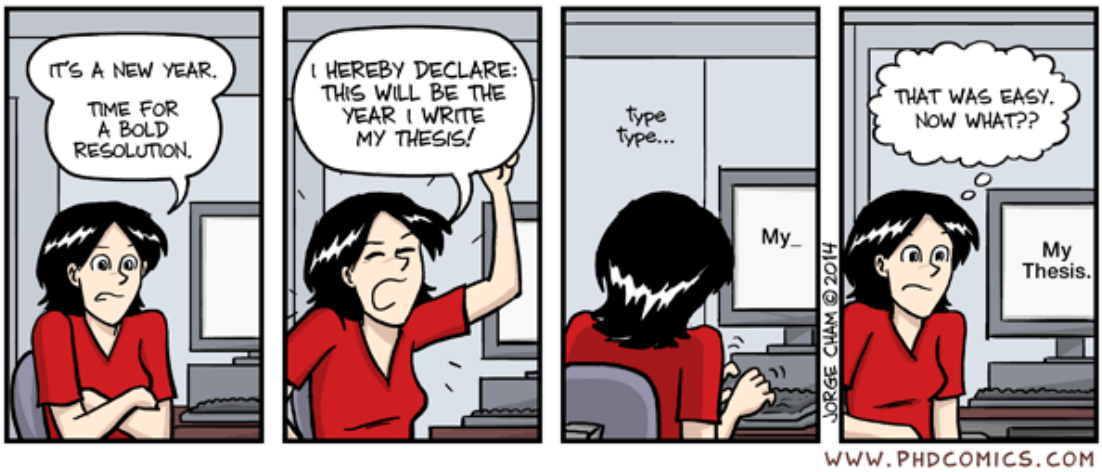
École doctorale et spécialité :

MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de Recherche :

Institut de Recherche en Informatique de Toulouse (UMR 5505)

Table des matières



Piled Higher and Deeper by Jorge Cham www.phdcomics.com

Today's Experiment...Succeeded

Chapitre 1

Introduction

In the beginning there was nothing, which exploded

Terry Pratchett

1.1 Impact énergétique des centres de calcul

Sans être au premier plan médiatique, les centres de calcul ont pris une place centrale dans nos vies. Les services tels que ceux fournis par Google, Facebook, Tweeter, Amazon, ..., reposent sur des grandes infrastructures de calcul et de stockage appelés centres de calcul. Pour fonctionner ils ont besoin de cette grande quantité de ressources qu'il a été nécessaire de regrouper afin de mutualiser les coûts de gestion et de refroidissement.

Au delà de ces grands services qui possèdent chacun plusieurs centaines de milliers de serveurs, une multitude de plus petits utilisateurs utilisent aussi ces centres de calcul, en co-location. Une petite mairie par exemple pourra plutôt utiliser une partie de serveur gérée par un opérateur de centre de calcul plutôt que de gérer en local, sans support informatique réel, son serveur web.

En 2014 déjà, les centres de calcul classiques pouvaient contenir plusieurs dizaines de milliers de serveurs sur des surfaces de l'ordre de $10000m^2$. Cette taille permet d'organiser de manière efficace la gestion des fluides (refroidissement, électricité). Pour un centre classique d'environ 40 000 serveurs utilisés par 500 000 services, la consommation électrique représente environ 10 mégawatts. Cette consommation est telle que le prix d'achat des serveurs est dépassé en quelques années par le prix de leur consommation électrique.

Avec l'explosion de l'utilisation de services sur internet, la consommation des centres de calcul explose. Entre 2000 et 2007, leur consommation annuelle au niveau mondial est passé de 70 à 330 térawatt-heures (TWh), et devrait atteindre 1000 à l'horizon 2020[47]. A l'échelle européenne, on estime que de 2007 à 2020, la consommation passera de 56 à 124 TWh par an[18]. En 2007 cette consommation représentait environ 2% des émissions de gaz carbonique (CO_2)[81], ce qui plaçait les centres de calcul au 6^{eme} rang mondial

si il s'agissait d'un pays[43]. Cette tendance à la hausse ne vas pas s'arrêter car les opérateurs de centres de calcul n'étaient que 8.5% à considérer en 2014 que la capacité de leur centre serait suffisante pour leurs besoins au delà de 2015[35].

La loi de Moore[74] qui prédit que la densité des circuits intégrés double tous les deux ans n'a pas permis de contrer cette augmentation de la consommation électrique. En effet, avec l'augmentation des densités des circuits intégrés, mais aussi l'augmentation des fréquences, les pertes électriques et la densité des flux de production de chaleur ont augmentés.

L'efficacité de ces centres de calcul est elle-même sujette à caution. En effet, une grande partie des serveurs qui y sont sont soit peu utilisés soit même devenus inutilisables. L'*Uptime Institute* évalue[95] à 20% les serveurs obsolètes, trop vieux, ou inutilisés. Quand à Google, une étude interne[4] a montré qu'en 2007 une grande majorité de leurs serveurs étaient utilisés à moins de 50% de leur capacité. Comme le fait même d'allumer un serveur consomme beaucoup d'électricité et produit de la chaleur qu'il faut extraire, ces chiffres montrent que sans une gestion très fine, les centres de calcul peuvent se révéler inefficaces.

1.2 Optimisation d'énergie

La gestion fine des centres de calcul est donc un enjeu majeur.

1.2.1 Positionnement

Il existe plusieurs niveaux où il est possible d'améliorer la qualité des centres de calcul :

Matériel Il s'agit d'améliorer l'efficacité des circuits intégrés[84, 85], leur agencement dans les serveurs[J2] [1], le matériel réseau[25], mais aussi de créer des moyens plus efficaces d'extraire la chaleur produite dans les serveurs[33]. La création de nouveaux matériels demande une grande expertise technique et la mise en place des améliorations peut se faire sur plusieurs années, en fonction du renouvellement des matériels déjà en place.

Applicatif Un des plus grands leviers est la réécriture des programmes de façon à les rendre plus rapide, moins consommateurs de ressources (mémoire, communication, stockage, ...). Dans certains cas cette ré-écriture est possible, parfois en s'aidant de modèles de programmation adaptés[J6]. Nous avons par exemple ajouté[C12] le contrôle de la fréquence des processeurs dans une application numérique de convection-diffusion, ce qui a permis de réduire la consommation énergétique de 20% sans modifier notablement la performance. Mais il n'est pas raisonnable de considérer que l'ensemble des logiciels s'exécutant dans les centres de calcul vont être ré-écrits pour être plus efficaces. Il n'est possible de le faire

1. Les références de type JX, CX, ChX et RTX sont respectivement les articles de journaux, conférences, chapitres, rapports techniques auxquels j'ai contribué, et DX les mémoires de doctorat de mes étudiants ayant soutenus

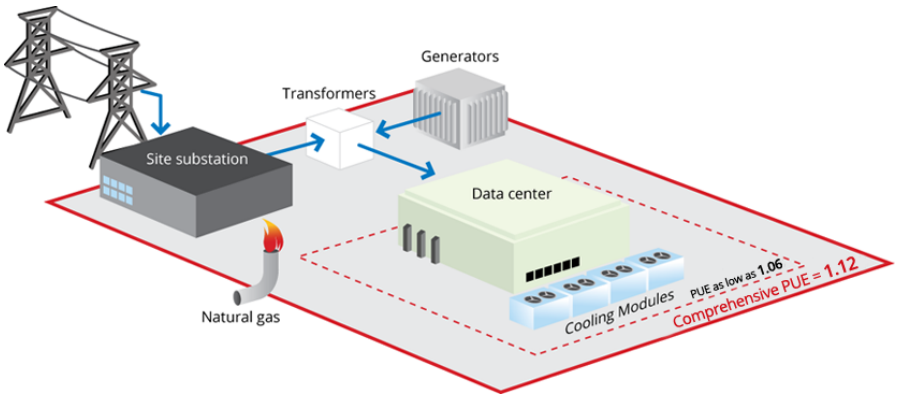


FIGURE 1.1 – Les serveurs d'un centre de calcul sont plongés dans un environement complexe, avec la gestion de l'électricité et du refroidissement (Crédits Google)

que ponctuellement, pour des programmes relativement simple et très souvent utilisés, donc pour lesquels le gain sera à la hauteur de l'investissement.

Intergiciel Il s'agit du logiciel qui gère le centre de calcul. Il décide quelle machine allumer ou éteindre, et où exécuter les applications par exemple. Une amélioration à ce niveau a l'avantage d'avoir un impact sur l'ensemble des opérations ayant lieu dans le centre de calcul, sans modification ni des applications ni des logiciels. Il permet donc de faire de la rénovation à moindre frais.

L'impact des améliorations au niveau matériel et applicatif sont les plus importants ponctuellement. En effet, changer les serveurs d'un centre de calcul pour des serveurs plus récents peut réduire drastiquement sa consommation électrique. Pourtant la possibilité d'utiliser les intergiciels dans l'ensemble des centres de calcul existants, avec un faible coût de déploiement, donne à ce niveau l'impact le plus important[19].

1.2.2 Quels effets de l'amélioration au niveau de l'intergiciel

Pour améliorer un centre de calcul, encore faut il parfaitement appréhender cet objet complexe. Un centre de calcul est composé de serveurs exécutant un certain nombre de tâches. Mais ces serveurs sont plongés dans un environnement complexe comme le montre la figure 1.1. Il faut bien évidemment apporter l'électricité jusqu'aux serveurs, mais comme ces derniers chauffent, il faut aussi alimenter et gérer des systèmes de refroidissement.

D'un point de vue puissance électrique, un centre de calcul n'utilise qu'une partie infime de l'énergie primaire produite du fait des pertes à toutes les étapes comme le montre la figure 1.2. Pourtant il s'agit d'un formidable effet de levier, car économiser quelques watts sur un serveur se traduira ainsi par ne pas avoir à produire près d'une centaine de watts.

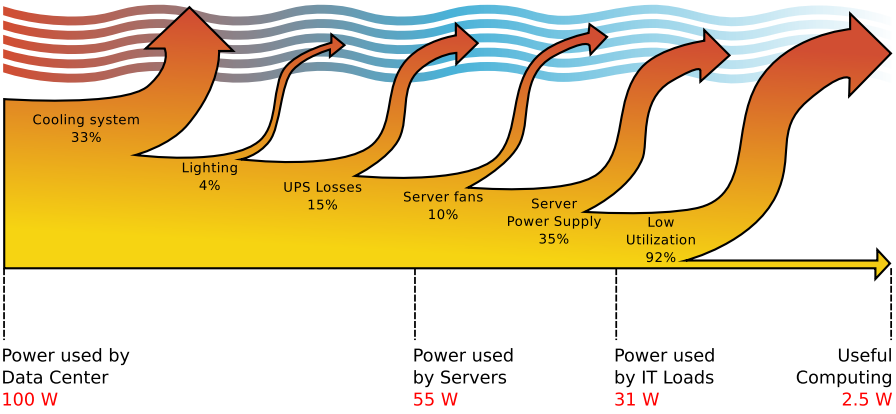


FIGURE 1.2 – Liste des différentes pertes de puissances entre la production de l'électricité et son utilisation dans un serveur. Données du *Rocky Mountain Institute*, 2008.

1.2.3 Intergiciel : Placement, déplacement et reconfigurations

Le rôle principal de l'intergiciel est donc l'orchestration d'un centre de calcul. C'est ainsi lui qui va décider quels serveurs vont être allumés ou éteints, où les tâches vont être placées. Mais il a aussi un rôle dynamique, il doit se rendre compte des problèmes (panne d'une machine par exemple) et les corriger. Pour chaque décision prise, il y a un large panel de possibilités. C'est de la qualité de ces décisions que va dépendre ensuite la qualité du centre de calcul.

De manière générale ces intergiciels utilisent une boucle autonome pour fonctionner (voir Figure 1.3). In s'agit d'un cycle où le système est surveillé (*Monitor*) grâce à des capteurs (*Sensor*). Les valeurs obtenues sont analysées (*Analyze*) et on en déduit des actions à effectuer (*Plan*). Les actions sont alors mises en place (*Execute*) grâce à des leviers (*Effector*). Une base de connaissance est manipulée par tous ces éléments comme base de la représentation du système et est donc utilisée pour prendre les décisions.

Pour la partie surveillance, le centre de calcul est instrumenté comme nous le détaillerons en section 3.1. L'intergiciel analyse ces données afin de savoir quelles sont les fautes, mais aussi les températures des serveurs ou les ressources consommées par les tâches s'exécutant sur eux (Section 2.2).

En utilisant ces données et un modèle mathématique du centre de calcul (la base de connaissance), l'intergiciel planifie les reconfigurations nécessaires à la bonne marche du centre de calcul, mais aussi à améliorer son fonctionnement (Section 5).

Enfin, l'intergiciel met en place sa décision, en utilisant les systèmes de gestion disponibles (tels que OpenStack ou OpenNebula décrits en section 4.3.1) qui eux même utilisent les leviers disponibles. On aura souvent des leviers tels que l'allumage ou l'extinction de serveurs, la migration de machine virtuelle, ou le changement de fréquence des processeurs par exemple (voir section 3.2).

Certaines décisions sont le résultat d'une boucle autonome locale aux serveurs, comme la gestion de la fréquence comme nous le décrirons en section 3.2.2.

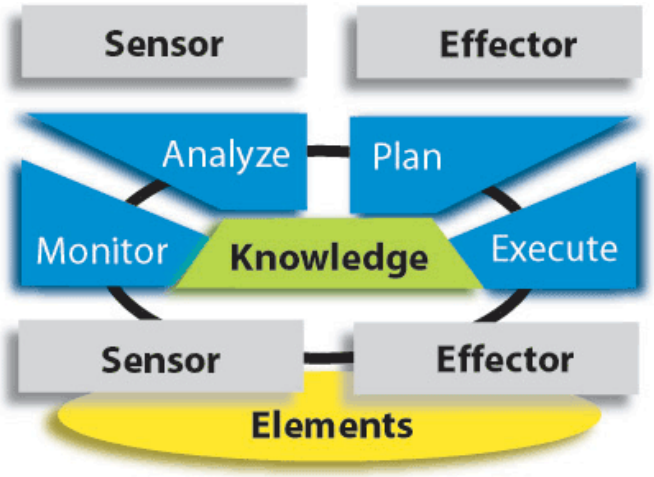


FIGURE 1.3 – Un gestionnaire de centre de calcul décide des reconfigurations en fonction de l'état du système, de ses connaissances, puis les met en place. (Crédits IBM)

1.3 Ultimate Question of Life, the Universe, and Everything

La question centrale à laquelle ce document va répondre est la suivante :

> **Objectif global**
> Comment gérer efficacement un centre de calcul ?

Pour répondre à cette question, un grand nombre de domaines sont à explorer. En effet il est nécessaire de résoudre les verrous scientifiques suivants :

- Que veut dire *efficace* et comment mesurer cette efficacité ?
- Que veut dire *gérer*, quelles sont les actions qu'il est possible de réaliser dans un centre de calcul et quels sont leurs impacts ?
- Comment vérifier théoriquement autant qu'expérimentalement que l'efficacité est atteinte ?
- Comment utiliser ces actions pour avoir un impact le plus efficace possible ?

1.3.1 Un fil rouge pour une réponse

Afin de répondre à cette question, il est nécessaire de l'approcher par plusieurs directions.

Ainsi pour répondre au premier point sur l'efficacité, un élément primordial concerne la mesure. En effet, comme nous l'avons montré dans la thèse de Leandro[D2] (co-encadrée avec Jean-Marc Pierson, 2011-2015), il ne suffit pas d'utiliser un wattmètre pour obtenir la consommation d'un programme, il faut tenir compte d'un certain nombre de biais et modéliser finement l'ensemble du serveur.

Une fois les méthodes de mesures stabilisées, reste la question épineuse du *quoi mesurer et pourquoi*. La question même de la mesure de la performance pure est à l'heure actuelle non résolue. Un opérateur de centre de calcul n'a pas connaissance des applications exécutées sur ses serveurs, il ne peut donc que difficilement évaluer leur performance. Même en ayant connaissance des applications, comparer la performance d'une application de bio-informatique avec celle d'un serveur web n'a pas grand sens. De manière plus générale faire l'évaluation énergétique d'un centre de calcul est particulièrement difficile si l'on tente de s'abstraire des cas particuliers. Ce fut l'un des objets principaux de participation en tant que responsable d'un *workpackage* sur les métriques et l'évaluation dans le projet européen CoolEmAll[J2] (2011-2014). Le but de ce projet était de modéliser, évaluer et améliorer les centres de calcul d'un point de vue performance, énergétique et thermique.

La modélisation des actions quand à elle est trop large pour pouvoir être résolue dans le court temps d'une thèse. Aussi les différents travaux sur l'optimisation ont, dans leur sillage, contribué aussi à améliorer la modélisation complète d'un centre de calcul.

La majorité des thèses que j'ai co-encadrées abordaient ce sujet de l'optimisation. Damien[D1] (2009-2013) a posé les bases de la problématique d'optimisation de l'énergie et de la performance. Tom[D3] (2011-2014) s'est plus particulièrement intéressé à la notion de Qualité de service, notion bien plus complexe que la simple performance. Thiam[D4] (2010-2014) s'est lui penché sur le problème de la coopération de plusieurs centres de calcul. Inès (2014-) explore l'utilisation d'énergies renouvelables pour alimenter les centres de calcul en réduisant leur empreinte écologique. Zong-Yi (2013-) quand à lui évalue comment transposer les méthodes de gestion de fréquence des serveurs dans le monde des protocoles réseaux bas niveau.

Les thèses dans lesquelles j'ai largement collaboré sont aussi sur ces thématiques. Violaine[J11] (2013-) s'intéresse aux possibilités offertes lors de l'utilisation de serveurs très hétérogènes. Landry[26] (2010-2013) s'est lui intéressé à l'identification des applications, des phases des serveurs au niveau système et à l'utilisation de cette identification pour améliorer l'efficacité énergétique des centres de calcul.

Cette vision et les différentes collaborations ont fortement été colorées par l'Action Cost IC0804 (2009-2013) sur *l'économie d'énergie dans les systèmes distribués à grande échelle*[2] à laquelle j'ai participé entre autre autour de la modélisation des centres de calcul en étant *working group leader* sur l'adaptation du matériel. Cette Action m'a permis d'échanger avec une grande partie de la communauté européenne intéressée par l'énergie dans les centres de calcul et a permis la mise en place du projet européen CoolEmAll. Mon implication dans l'Action Cost IC1305 Nesus (*Network for Sustainable Ultrascale Computing*, 2013-) en tant que *working group leader* sur les modèles de programmation et intergiciels pour le calcul ultrascale participe de la même volonté d'interaction et d'ouverture à la communauté.

Ces travaux ont fait l'objet de publications dans 33 conférences et de 11 articles de journaux internationaux avec comité de relecture ainsi que de 9 chapitres d'ouvrages de synthèse.

2. `http://www.irit.fr/cost804`

1.3.2 Centre de calcul : Un grand outil scientifique

Expérimenter in vivo est toujours complexe. Dans toutes les disciplines, il est nécessaire de faire attention aux biais d'expérimentation et d'interprétation[56]. Il est aussi souvent complexe de mettre en place les expérimentations qui, une fois lancées, peuvent se révéler être très longues.

L'ensemble du travail présenté ici n'aurait jamais vu le jour sans la plateforme Grid'5000[14]. En effet, l'objet d'expérimentation est ici le centre de calcul. Pourtant il est difficile d'aller voir un opérateur de centre de calcul et de lui demander de s'arrêter quelques jours, le temps de faire une expérience.

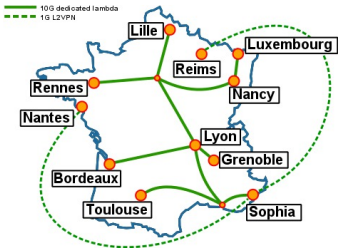


FIGURE 1.4 – Les 11 sites de Grid'5000

Grid'5000 fournit à la communauté française un grand nombre de ressources (1000 serveurs, 8000 coeurs, 11 sites) dédiées à l'expérimentation en informatique (Figure 1.4). Des travaux récents ont permis de rajouter des wattmètres dans une majorité de sites, rendant cet outil parfait pour des tests liés à l'économie d'énergie.

En complément à la majorité des grandes masses de ressources de calcul ou de stockage qui sont au service de la recherche physique, mathématique, ..., Grid'5000 est un formidable outil pour la recherche en informatique.

1.3.3 Une approche holistique par simulation

Malgré tout, ne faire que des tests grandeur nature est presque impossible. La simulation, au prix d'une réduction de la précision, permet d'obtenir des résultats raisonnablement rapidement. La précision de ces résultats est grandement liée à la qualité des modèles utilisés dans le coeur du simulateur.

Il est donc nécessaire d'avoir une modélisation fine, autant de la consommation de ressources que de la dissipation thermique. Nous nous sommes impliqués dans le projet CoolEmAll avec cette vision globale. Dans ce projet, les simulations utilisent des modèles de processeurs et d'applications mais aussi un modèle thermique basé sur la dynamique des fluides au niveau du centre de calcul dans sa globalité.

Un des résultats de ce projet est un outil de simulation qui permet de tester dans des conditions les plus réalistes possibles des méthodes de gestion de centre de calcul, d'évaluer leur impact, avant de passer à un cadre plus réel.

1.4 Plan

Dans la suite, nous partirons des briques de base, des éléments de modélisation pour ensuite montrer leur utilisation afin de gérer efficacement un centre de calcul.

En plus de l'introduction, et du chapitre concluant ce mémoire avec un bilan et des perspectives, la structure sera la suivante :

Chapitre 2 : Modélisation et évaluation de centres de calcul Dans ce chapitre nous expliciterons et définirons précisément ce qu'est un centre de calcul ainsi que ses principales caractéristiques. Nous répondrons aussi à la question de ce qu'est que l'*efficacité* dans le cadre de ce mémoire.

Chapitre 3 : Connaissance et degrés de liberté Ici seront abordées les différentes façons d'obtenir des informations sur un centre de calcul, ainsi que les leviers utilisables pour en modifier l'état et leur impact.

Chapitre 4 : Les outils de l'évaluation de performance Il sera alors temps d'aborder les outils de résolution exacts, de simulation et les méthodes d'expérimentation nécessaires pour évaluer la qualité des systèmes proposés.

Chapitre 5 : Résolution approchée par heuristiques Dans ce chapitre, nous allons étudier le comportement des heuristiques de résolution du problème, ainsi que leurs caractéristiques et l'impact de ces caractéristiques sur leurs effets.

Chapitre 6 : Vers une vision distribuée Enfin nous allons étendre ces résultats aux fédérations de centres de calcul. Ce passage à l'échelle se traduira aussi pas une plus grande autonomie des serveurs individuels.

Chapitre 2

Modélisation et évaluation de centres de calcul

> There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable. There is another theory which states that this has already happened
>
> Douglas Adams

Dans un premier temps il semble nécessaire de préciser le modèle utilisé pour les centres de calcul. En effet il existe plusieurs points de vue, suivant qu'il s'agisse d'en fabriquer, exploiter ou optimiser un.

Dans la suite on s'intéressera plus particulièrement aux centres de calcul en tant que tels, fournissant leurs ressources à des utilisateurs externes ou à ceux des très grandes entreprises, fonctionnant sur le même principe. Les centres de calcul professionnels dédiés ou les centres de calcul des grandes entreprises sont de l'ordre de grandeur de 50 000 à 100 000 serveurs consommant de 10 à 30MW et d'une superficie de l'ordre de 10 000 m^2. Le périmètre de cette étude se limitera aux centres de calcul, sans tenir compte de leur environnement. Par exemple la réutilisation de la chaleur ou l'utilisation d'autres étages d'un bâtiment pour des bureaux ne seront pas abordés ici.

2.1 Modèle haut niveau d'un centre de calcul

2.1.1 Un centre de calcul

Il existe plusieurs niveaux de détails pour modéliser un centre de calcul. Les niveaux les plus précis permettent une estimation très fine de leur comportement mais au prix d'une grande complexité. Est-il par exemple pertinent de modéliser la consommation électrique liée à l'allumage d'une salle lorsqu'un opérateur vient changer un composant

défectueux ? De même est-il utile de modéliser finement la production thermique dans un cadre de *free-cooling* (refroidissement passif en utilisant de l'air extérieur à température ambiante) ?

À haut niveau, toute modélisation de centre de calcul a besoin d'intégrer des règles de cohérences permettant la modélisation minimale du monde physique. On considère ici avoir des serveurs regroupés dans un centre de calcul. Sur les serveurs vont s'exécuter des applications, souvent des services, parfois encapsulées dans des machines virtuelles. Ces applications vont avoir des besoins de ressources de deux types, fluides et rigides[97]. Si une application n'a pas assez de ressources rigides, elle ne pourra pas s'exécuter, comme par exemple dans le cas de la mémoire. Dans le cas où une application n'a pas assez de ressources fluides, comme de la bande passante, alors elle pourra s'exécuter mais de manière dégradée. Dans la suite, pour alléger la rédaction, nous allons utiliser de manière interchangeable les termes *machine virtuelle*, *application* et *service*.

On a alors à un **instant précis** les caractéristiques suivantes dans le cas d'un centre de calcul n'exécutant que des machines virtuelles :

Une machine est soit allumée, soit éteinte (2.1)

Une machine virtuelle (VM) est localisée sur une seule machine (2.2)

Une VM est soit stoppée soit en train de s'exécuter (2.3)

Une VM sur une machine non allumée est stoppée (2.4)

Une machine exécutant une VM est allumée (2.5)

L'ensemble des VM consomment au plus les ressources de leur machine (2.6)

Une VM s'exécutant reçoit exactement les ressources rigides demandées (2.7)

Une VM recevant moins de ressources fluides que demandées est ralentie (2.8)

Viennent ensuite les contraintes **temporelles** :

Une machine peut s'éteindre (/s'allumer) que si elle était allumée (/éteinte) (2.9)

Une VM ne peut migrer qu'entre deux machines allumées (2.10)

Cette modélisation relativement simple est une modélisation de type particulaire physique. Il s'agit d'un ensemble de contraintes que doivent suivre toutes machines physiques et virtuelles dans un centre de calcul. Ainsi de manière générale, tout simulateur ou tout modèle mathématique doit implémenter l'ensemble de ces règles pour être considéré comme valide.

Pourtant loin d'être absolu cet ensemble de règles est parfois à panacher en fonction des besoins. Par exemple, l'ajout du temps (deuxième jeux de règles) complexifie grandement le problème et l'on va parfois enlever cette dimension pour se retrouver dans un cadre plus simple. Ainsi nos premiers travaux[J1] sur ce type de placements reposent uniquement pour des soucis de simplification sur les règles 2.1 à 2.8 et portent donc uniquement sur le placement de VM sans considération de temps. Ces travaux ont été initiés lors de visites d'Henri Casanova, professeur à l'université de Manoa à Hawai (USA) pendant un mois en 2008 et un mois en 2010.

Mais d'autres simplifications des règles peuvent avoir lieux dans d'autres cadres. Par exemple dans [C7] avec des collègues mathématiciens de l'IMT (Institut Mathématique de Toulouse) et de l'ONERA Toulouse nous avons relâché la règle 2.2 dans un cadre de grande échelle. Sans cette règle, une VMs peut être à moitié sur un serveur, et à moitié sur un autre. Pour un système avec peu de VMs, l'impact relatif est grand, mais dans un environnement de plusieurs centaines de milliers de machines l'erreur relative devient insignifiante. Cette approximation permet de traiter des "masses" de VMs et donc de réduire la complexité en calcul des allocations. Précise pour une fédération de centres de calcul telle que celle d'Amazon, cette hypothèse est totalement irréaliste dans un cadre de petits centres de calcul.

2.1.2 Modèle thermique et énergétique

La description précédente montre la modélisation du coeur du centre de calcul, l'état des serveurs et des machines virtuelles. De ces données découlent un certain nombre d'éléments : la température des serveurs ainsi que leur consommation électrique.

2.1.2.1 Consommation des serveurs

Le modèle classique[87] de consommation d'une machine dans un centre de calcul est obtenu par la formule suivante :

$$P = P_{idle} + (P_{max} - P_{idle})Load \tag{2.11}$$

Dans cette formule, P_{idle} et P_{max} représentent respectivement la puissance à vide et maximum d'un serveur. *Load* représenta la charge du serveur, avec une valeur comprise entre 0 et 1.

Avec l'utilisation omniprésente des processeurs capable d'adapter leur fréquence, plusieurs études[15, 6, 76] ont rajouté la fréquence. En effet, en fonction de la fréquence du processeur, son voltage va devoir changer. Plus la fréquence est haute, plus le voltage est haut et alors plus la consommation augmente :

$$P = P_{idle} + Load \times Capacitance \times V_{cpu}^2 Freq \tag{2.12}$$

Ici *Capacitance* représente l'efficacité énergétique du processeur et est lié à la technologie CMOS sous-jacente. V_{cpu} est le voltage du processeur, et $Freq$ sa fréquence.

Le dernier élément habituellement rajouté[89] par rapport au processeur est sa température. En effet les pertes électriques dans la technologie CMOS augmentent avec la température. Cette perte dépend du processeur (θ_{cpu}) :

$$P = P_{idle} + Load \times Capacitance \times V_{cpu}^2 Freq + \theta_{cpu} T_{cpu} \tag{2.13}$$

Cette famille de modèle modélise uniquement le processeur car il est souvent considéré qu'il s'agit de la partie la plus consommatrice, ou tout au moins, la partie la plus

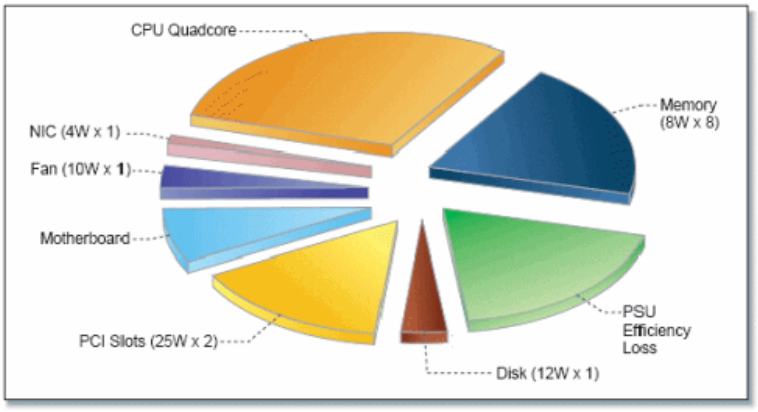


FIGURE 2.1 – Consommation des sous-systèmes d'un serveur PC classique avec un processeur Intel Xeon quadricoeur (Crédits Intel labs 2008).

changeante de la consommation. Même si la consommation d'un processeur est prépondérante dans un serveur, elle est loin d'écraser les autres consommations comme le montre la Figure 2.1. Même pour la partie dynamique, nous avons montré lors d'une collaboration avec le professeur de l'Université Technique de Vienne Helmut Hlavacs[C9] lors d'une visite d'un mois que j'ai réalisée dans son université en 2010 que ne prendre en compte que le processeur menait à des imprécisions de 5 à 10%.

D'autres approches rajoutent encore un certain nombre d'éléments aux modèles. Ainsi différents travaux[34, 29, 50] présentent les modèles de la mémoire, du disque, du réseau et enfin du processeur et utilisent un modèle global additionnant ces modèles indépendants. PowerAPI[76] utilise ce principe en extrayant les constantes de son modèle des documents de références des constructeurs par exemple.

Une autre approche[C9, D2, 70] consiste à ne pas avoir à priori un modèle mais à utiliser des méthodes d'apprentissage afin de créer le modèle adapté à une machine particulière. Les méthodes de régression linéaires et les réseaux de neurones sont parmi les méthodes les plus utilisées. La difficulté de ces méthodes vient de la nécessité d'une phase d'apprentissage. Dans le cas d'un centre de calcul où l'hétérogénéité est habituellement limitée, il est nécessaire de faire l'apprentissage sur un représentant de chaque type de serveur, ce qui réduit ce coût. L'utilisation des méthodes par apprentissage pour obtenir un modèle de consommation électrique de serveur sera explicité en Section 3.1.3.4.

Il s'agit de trouver un compromis entre la précision, le coût d'utilisation, mais aussi le coût de création. Ainsi si il s'agit d'être capable de faire la distinction entre plusieurs serveurs identiques dans un centre de calcul, tenir compte de la température fait sens. Dans un autre cas, où toutes les machines sont très différentes, même le plus simple modèle peut être suffisant car discriminant. Par contre, pour avoir un modèle très fin, surtout dans le cas où l'on connaît les applications qui s'exécuteront, les méthodes par apprentissage permettent de prendre en compte l'ensemble du serveur de manière sûre. Il devient ainsi facile de prendre en compte l'allumage/extinction des ventilateurs par

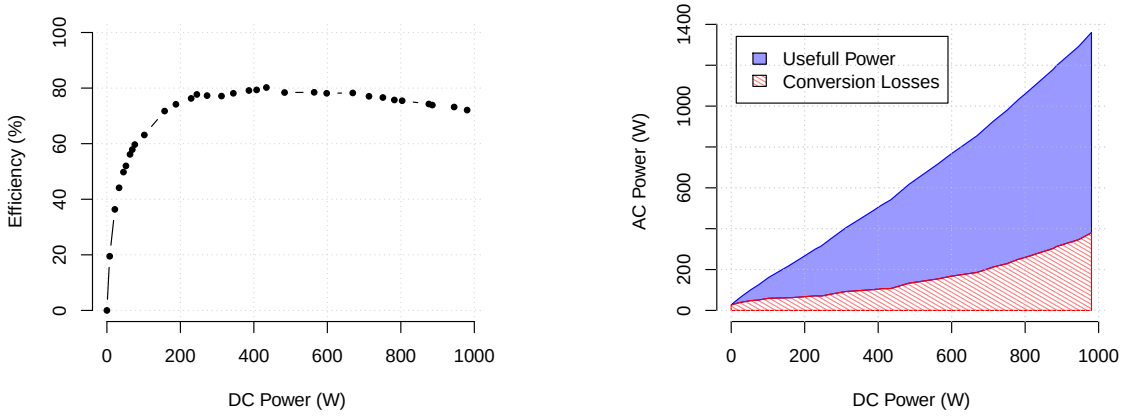


FIGURE 2.2 – Profil d'efficacité (gauche) et pertes de conversion (droite) d'une alimentation

exemple.

Enfin, de manière transverse à tous les modèles, il est nécessaire de tenir compte de l'ensemble du serveur. Les ventilateurs par exemple peuvent être alimentés par la carte mère et peuvent contribuer de manière assez forte à la consommation du serveur, avec des pics jusqu'à 20%[59] pour des systèmes de type *Blade*. Un des biais est souvent l'oubli de modéliser le bloc d'alimentation. Un exemple des pertes provoquées par l'alimentation est montré en Figure 2.2. A gauche, on peut voir que pour l'alimentation que nous avons mesurée, l'efficacité monte très vite jusqu'à un plateau à 80% puis diminue. Dans des travaux où l'alimentation va travailler entre 200 et 700W en sortie on peut considérer que son efficacité est constante, mais dans les autres cas, il est nécessaire de tenir compte de son profil complet d'efficacité. Dans [D2] nous avons montré qu'une modélisation polynomiale permet de les modéliser. Ce raffinement de la modélisation nous a ensuite permis d'améliorer un modèle de consommation global et d'en réduire l'erreur de 20%.

2.1.2.2 Gestion thermale des serveurs

Un exemple simple de phénomène complexe vient de l'impact de la production de chaleur des serveurs sur leurs voisins. Ainsi, un serveur consommera d'autant plus que d'autres serveurs proches sont chauds. À travail strictement égal, et donc à consommation de ressources égalles, sa consommation électrique sera différente. Pour modéliser ce phénomène il est possible d'utiliser une matrice d'influence[99, 100, J9, C17] appelée *D-matrix* (*Distribution matrix*) qui explicite pour chaque couple de machine l'impact de la température de l'une sur l'augmentation de la température de l'autre. Dans [Ch4] nous avons par exemple mesuré pour un système de serveurs haute densité un impact de 2 à 2.5°C lorsque un serveur à pleine charge jouxte un autre.

Quand à la production de chaleur venant des composants internes, elle impacte la température en tenant compte des caractéristiques des matériaux utilisés. Le modèle

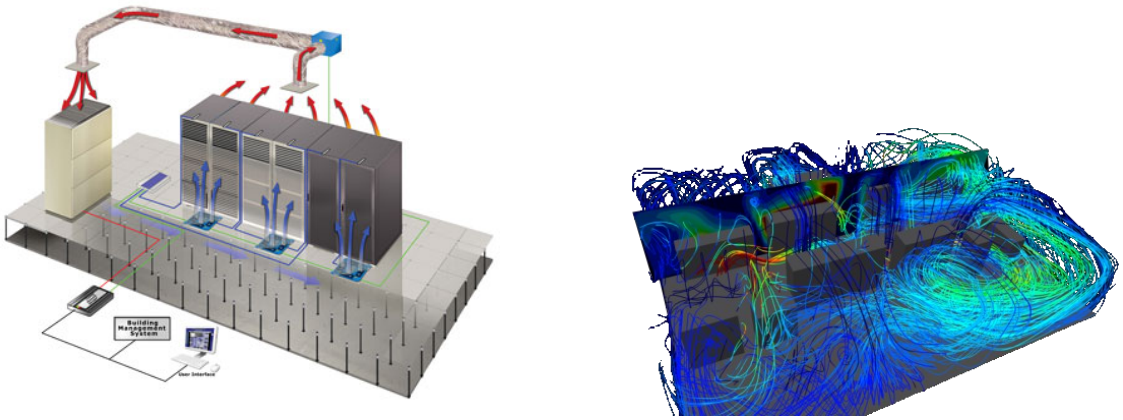


FIGURE 2.3 – Schéma théorique de l'architecture d'un refroidissement par air à gauche (crédits Electronic Environments Corporation), et modélisation des flux d'air à droite.

concernant le processeur[J12, 108, 57] (hors impact des autres serveurs) est le suivant :

$$T_{cpu}(t+dt) = T^{\infty}_{cpu} - (T^{\infty}_{cpu} - T_{cpu}(t)) * e^{-dt/sink} \tag{2.14}$$

$$T^{\infty}_{cpu} = T_{idle} + Load * (T_{max} - T_{idle}) - dT_{fan} * \mathbb{1}_{(P_{Fan} \neq 0)} \tag{2.15}$$

Ici T^{∞}_{cpu} représente la température d'équilibre qui serait atteinte si la charge ($Load$) reste constante. On utilise ici le modèle linéaire simple (Équation 2.11). $sink$ représente la capacité thermique du radiateur, T_{idle} et T_{max} les températures du processeur à vide et à pleine charge. dT_{fan} et P_{fan} représentent respectivement l'impact sur la température du ventilateur si il est présent et sa consommation de puissance.

Enfin, la température de sortie du fluide de refroidissement (air ou eau le plus souvent) :

$$T_{out} = P_{server}/K + T_{in} \tag{2.16}$$

Avec K la capacité d'absorption thermique. Par exemple pour l'air cette capacité est $\rho V C_p$, avec ρ la densité de l'air, V le volume et C_p la capacité thermique de l'air.

2.1.2.3 Consommation de l'infrastructure

D'un point de vue de l'infrastructure le fluide de refroidissement poursuit un cycle sans fin. Il est chauffé dans le serveur, puis envoyé dans un système de refroidissement. Une fois refroidi il retourne vers le serveur (Figure 2.3).

Cette vision théorique est relativement réaliste dans le cas du refroidissement par eau mais moins réaliste dans le cas d'un refroidissement par air. La figure 2.3 montre une

simulation de la circulation des flux d'airs dans le cadre du projet CoolEmAll. Dans ce cas, une partie de l'air chauffé n'est pas extrait et retourne directement vers les serveurs. Il s'agit du phénomène décrit dans la section précédente et modélisé par la *D-matrix*.

Pour ce qui est de l'efficacité de la partie refroidissement, elle dépend du type de refroidissement ainsi que du modèle exact de matériel utilisé.

Par exemple, dans le cas de l'air[75], on peut avoir :

$$P_{ac} = P_{server}/COP \tag{2.17}$$

$$COP = (0.0068 * T_{sup}^2 + 0.0008 * T_{sup} + 0.458) \tag{2.18}$$

2.1.3 Description unifiée

Dans les sections précédentes, nous avons explicité un certain nombre de modèles. Ils ont en commun d'être basés sur des constantes dépendant du matériel utilisé. Ces constantes viennent d'un grand nombre de domaines différents : performance, électricité, thermique.

Dans le projet CoolEmAll[J2], pour résoudre le problème de la description de centres de calcul, nous avons mis en place un format de fichier d'échange de description d'infrastructure : *Data Center Efficiency Building Block* (DEBB)[9].

Ce format d'échange comporte pour chaque élément des informations le définissant complètement comme le montre la figure 2.4

Il comporte plusieurs sections. Une section décrit les matériels (serveur, refroidissement, réseau), une autre section décrit leur position dans l'espace mais aussi leur interconnexion réseau et leur modèle 3D.

En utilisant le DEBB décrivant un centre de calcul, il est donc possible de faire des simulations de gestion de ressources et de VM, d'extraire des métriques telles que l'efficacité énergétique, mais aussi de faire des simulations de mécanique des fluides décrivant les flux d'air.

2.2 Les applications

Une fois que le modèle physique décrit les comportements de chacun des éléments que sont les machines physiques et virtuelles, il est nécessaire d'avoir des modèles d'impact : comment une VM se comporte si elle a moins de ressources que demandées ? Combien d'électricité consomme cette VM ? Mais aussi comment la température d'une machine évolue au cours du temps ?

2.2.1 Consommation de ressources

Afin d'effectuer un travail, une application va devoir consommer des ressources. Une des problématiques concernant la connaissance d'une application est qu'elle est indissociable de son contexte : matériels sur laquelle elle s'exécute, autres applications avec

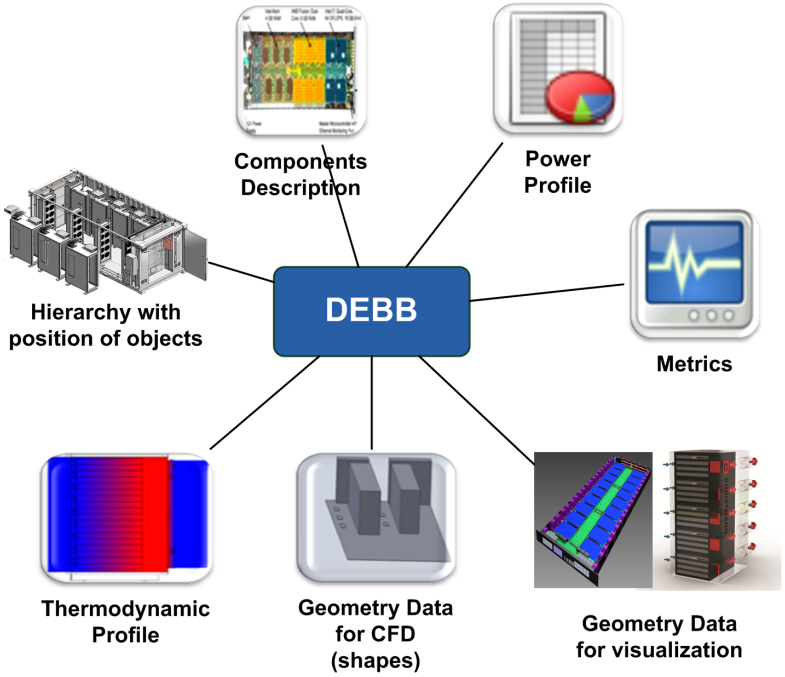


FIGURE 2.4 – Contenu d'un fichier DEBB décrivant un centre de calcul

lesquelles elle communique, mais aussi autres applications avec lesquelles elle partage ses ressources.

Dans ce cadre nous allons considérer qu'une application s'exécutant seule sur un matériel particulier peut être définie comme une suite de phases. Les phases sont définies comme des durées pendant lesquelles la consommation de ressources est presque constante. Chaque phase est définie par sa longueur (secondes), les ressources (dont l'énergie) consommée. Un exemple est donnée en figure 2.5 pour l'application de lancer de rayons *c-ray*. Il montre que cette application est composée d'une première étape (entre 0 et 180s) pendant laquelle les ressources mémoires augmentent et les ressources processeurs sont utilisées fortement. L'étape suivante (entre 180 et 230) voit la stabilisation de l'allocation mémoire et une baisse de l'utilisation du processeur. Enfin, la dernière étape montre une utilisation du disque. Ces trois étapes sont modélisées ici par un grand nombre (22) de phases assez courtes (une dizaine de secondes environ). Pendant chacune de ces phases, le comportement de l'application peut être considéré comme stable.

2.2.1.1 Modèle de puissance de calcul mono-application

Pour être capable d'évaluer le comportement d'un centre de calcul, il est nécessaire dans un premier temps de comprendre combien de temps une VM met à s'exécuter et quelles ressources elle consomme à un instant donné.

Dans le cadre de nos travaux, on considère deux types de ressources, les ressources fluides et les ressources rigides. Une ressource rigide (telle que la mémoire) est une res-

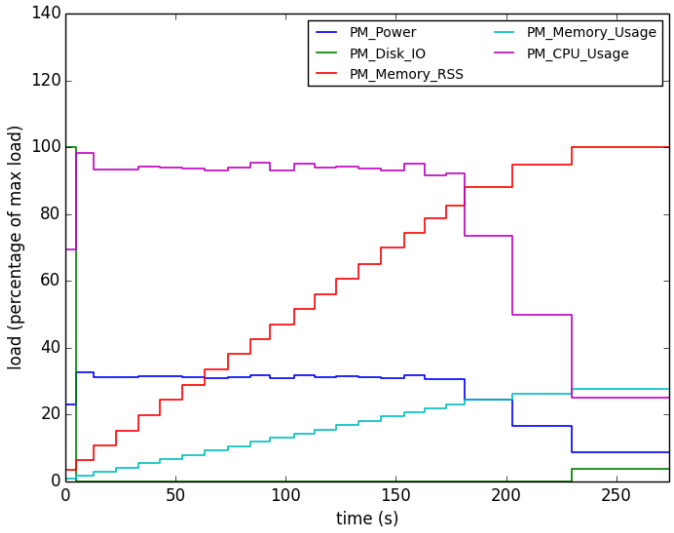


FIGURE 2.5 – Exemple de profil d'application : c-ray (lancer de rayon)

source totalement nécessaire. Si la quantité de ressource n'est pas disponible, alors la VM ne peut pas s'exécuter. Dans le cas d'une ressource fluide (telle que la bande passante) si la quantité demandée n'est pas disponible, la VM peut quand même s'exécuter mais plus lentement (voir Figure 2.6).

Plus précisément lorsqu'une VM s'exécute seule sur une machine, on peut évaluer son ralentissement en fonction de contraintes de ressources : DVFS (changement de fréquence du processeur), ALR (changement de bande passante de la carte réseau). Pour chacune des ressources fluides, on évalue la quantité de ressources demandée (X_f) et celle fournie Y_f. On aura le ralentissement suivant :

$$max(1, \frac{X_f}{Y_f}) \tag{2.19}$$

Lorsqu'on a plusieurs ressources fluides, le plus grand ralentissement est celui qui s'applique.

Pour évaluer l'impact du DVFS par exemple, on considère qu'on a simplement un changement proportionnel en fonction de la fréquence :

$$max(1, \frac{X_f}{Y_f} \frac{Freq_{max}}{Freq_{current}}) \tag{2.20}$$

Le plus grand problème dans cette formulation est qu'il est nécessaire d'avoir X_f et donc dans un cadre hétérogène de connaître la consommation de ressources de chaque application sur chaque architecture.

Pour résoudre ce problème, nous avons développé un convertisseur de profils d'applications (*Application Profile Converter*) dans le cadre du projet européen CoolEmAll.

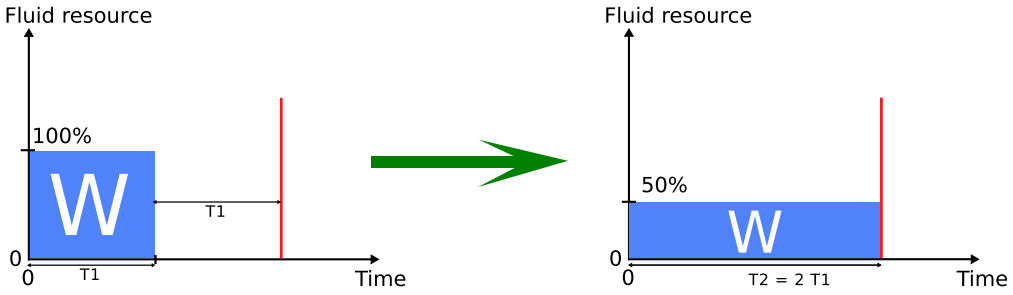


FIGURE 2.6 – Allouer moins de ressources fluides à une tâche entraîne un ralentissement

Une description complète est disponible dans le livrable D4.8[RT1] mais le principe est néanmoins relativement simple. On considère avoir un profil de consommation de ressources rigides et fluides d'une application sur une architecture donnée. Grâce à ce profil, on détermine quelle est la ressource limitante (goulet d'étranglement). Ce profil de consommation est traduit sur le profil du serveur cible, en tenant compte du goulet qui peut changer du fait des différences entre les serveurs. Par exemple, il est possible qu'une application ayant comme limite le processeur sur une machine ait le réseau à la place après une migration sur une machine avec un processeur beaucoup plus rapide et/ou un réseau beaucoup plus lent.

Cette technique est une généralisation des techniques de changement de fréquences décrites ci-dessus au cas de machines complètement différentes. La grande limitation de cette technique est que lorsque les architectures sont profondément différentes (entre un processeur ARM et un GPU par exemple) la règle de proportionnalité ne s'applique plus.

2.2.1.2 Modèle de puissance de calcul multi-applications

Dans le cas de multiples applications le parti-pris principal consiste à considérer que la somme des applications consomme la somme des ressources de chaque application. Si à très gros grain cette affirmation est vrai, dans la réalité deux applications ayant des goulets d'étranglements très différents auront un ralentissement moindre. Par exemple une application limitée par le processeur et une application limitée par la mémoire se ralentiront moins que deux applications limitées par leur accès à la mémoire. La figure 2.7 obtenue sur un quadri-coeur illustre ce principe. 4 applications limitées par le processeur ou 3 et une limitée par la mémoire n'interagissent pas et ne se ralentissent pas les unes les autres. Lorsqu'au moins deux applications ont comme goulet d'étranglement la mémoire, elles sont ralenties.

Comme dans la réalité on aura finalement un ralentissement moindre lorsque les applications sont limitées par des goulets différents, il est d'usage de considérer l'hypothèse conservatrice qui est qu'il s'agit du même. Dans le cas d'une ressource fluide, on aura selon le même principe que pour le DVFS un ralentissement lorsque deux applications s'exécutent sur une même machine en demandant plus de ressources que disponibles. Il existe plusieurs méthodes pour partager les ressources. La première arrivée peut ne pas être ralentie, contrairement à la seconde. Selon l'usage habituel en système, on va le plus

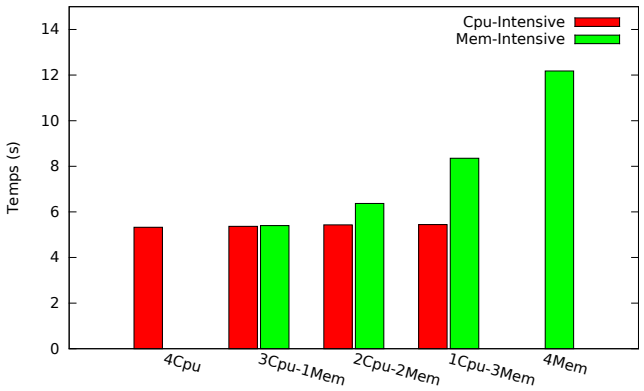


FIGURE 2.7 – Temps d'exécution en fonction du panachage d'applications concurrentes sur un processeur quadri-coeur. Les applications *Mem-intensive* et *Cpu-intensive* ont la même durée lorsqu'elle sont exécutées seules.

souvent répartir équitablement le ralentissement. Si les ressources demandées sont X_f^1 et X_f^2, alors on aura pour les ressources fournies équitablement les Y_f^a avec $a = 1$ ou $a = 2$:

$$Y_f^a = X_f^a \times min(1, \frac{1}{X_f^1 + X_f^2}) \tag{2.21}$$

La modélisation fine au niveau système et architecture des interactions entre applications est une piste intéressante de recherche.

2.2.2 Énergie des applications

Un élément particulièrement complexe de la gestion d'un centre de calcul est lié à la virtualisation. Ce principe permet de s'abstraire des machines physiques et de n'avoir plus qu'une vision logique à base de machines virtuelles. Pour le gestionnaire d'infrastructures, savoir combien chaque machine consomme est important. Pourtant, pour l'utilisateur, qui lui gère son "parc" de machines virtuelles, la donnée intéressante est de savoir combien consomment ses machines virtuelles et non pas les machines physiques qu'il partage avec d'autres utilisateurs. Le problème principal est de définir quel est le sens de cette consommation. En effet il n'existe pas de système physique tel qu'un wattmètre capable d'évaluer la consommation d'une application parmi d'autre sur un serveur.

Pourtant cette valeur est importante tout autant directement qu'indirectement. Directement car un gestionnaire de centre de calcul peut vouloir facturer ses utilisateurs en fonction de leur consommation, l'électricité étant la plus grande partie de ses frais de fonctionnement. Indirectement car connaître la consommation d'un service tournant dans une machine virtuelle permet de comparer plusieurs services équivalents, mais aussi permet aux utilisateurs de ces services d'en connaître l'impact, ce qui contribue à leur sensibilisation.

Dans [D2, J4] nous avons défini une méthodologie qui permet non seulement dévaluer au niveau système la consommation (voir Section 2.1.2.1) d'une machine, mais aussi au niveau d'une application.

Pour pouvoir estimer la consommation d'une application, il est possible d'utiliser le même modèle que celui d'un serveur complet mais en ne gardant que les valeurs liées à une application particulière. Pour le modèle affine basé sur la charge par exemple (Formule 2.11), il suffit d'utiliser la charge du processus au lieu de la charge totale de la machine pour estimer la partie dynamique.

Pour répartir la partie statique (P_{min} dans le cas de la Formule 2.11) la méthode choisie est de le faire selon le même ratio que la partie dynamique. Si une application consomme deux fois plus pour la partie dynamique, par convention on va lui attribuer deux fois plus pour la partie statique. Cette convention est liée au cadre d'utilisation et sera donc à adapter à chaque fois en fonction du but recherché : sensibilisation, optimisation, facturation, impact environemental, ...

Pour pouvoir généraliser un modèle de consommation du niveau système au niveau applicatif sans modification, simplement en transposant les variables du niveau système au niveau applicatif, nous avons prouvé[D2] qu'il est nécessaire que ce modèle soit linéaire.

2.3 Techniques d'évaluation classique

Dans le début de ce chapitre nous avons vu un certain nombre de modèles permettant d'évaluer finement la performance d'une application, autant d'un point de vue énergétique que d'un point de vue performance.

Mais l'évaluation d'un centre de calcul est plus qu'une agrégation de valeurs provenant de mesures ou de modèles. Une évaluation est faite dans un but. Par exemple on peut chercher les améliorations possibles pour consommer moins d'énergie dans un centre de calcul ou chiffrer l'amélioration produite par des changements de politique de gestion. Les valeurs étudiées seront différentes dans les deux cas.

Historiquement les centres de calcul avaient une vocation scientifique et souvent académique. La métrique la plus utilisée était alors le *makespan*, c'est à dire le temps entre le début et la fin d'un ensemble de tâches de calcul. Les applications, même complexes étaient exécutées sur du matériel dédié, sans partage de serveurs avec d'autres applications. Dans un cadre légèrement différent, les super-calculateurs sont quand à eux définis par leur capacité maximale à faire du calcul, comme en témoigne l'importance du top500[31] dans la communauté du calcul haute performance.

Mais depuis l'avènement de l'informatique en nuage, la situation s'est fortement complexifiée, le matériel est partagé et les applications sont devenues pour beaucoup des services. Comparativement à une application, un service n'a pas forcément de fin, mais a une consommation de ressources dépendant des utilisateurs et donc du temps. Dans ce cadre il devient difficile de définir la performance d'une application ou d'un centre de calcul.

1.	E_{DC}	Total Energy Consumption
2.	PUE_3	Power Usage Effectiveness – level 3
3.	PUE_4	Power Usage Effectiveness – level 4
4.	EWR	Energy Wasted Ratio
5.	$Prod$	Productivity
6.	$RCI_{LO,HI}$	Rack Cooling Index
7.	IoT	Imbalance of CPU temperature
8.	$CAPEX$	Capital Expenditure
9.	$OPEX$	Operation Expenditure or Electricity Costs
10.	CO_2	Carbon Emissions (embedded and power consumption)

TABLE 2.1 – Métriques à destination des opérateurs de centre de calcul

Pour tenter de répondre à cette question, des entreprises proposent des panneaux de pilotages de centres de calcul où sont regroupés les informations les plus intéressantes. Ainsi *Future Facilities*[1] propose une description basée sur trois valeurs : *Capacity*, *Availability* et *Efficiency* (Figure 2.8).

De manière générale, les KPIs (*Key Performance Indicators* ou indicateurs clefs de performance) sont censés être des indicateurs permettant la prise de décision efficace. Ainsi les trois valeurs précédentes sont considérées comme des KPIs.

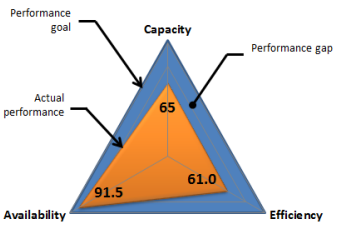


FIGURE 2.8 – Description visuelle de la performance d'un centre de calcul. ©*Future Facilities*

2.3.1 Métriques classiques des centres de calcul

Une de nos contributions au projet Européen CoolEmAll a été la définition[RT8] d'un ensemble de 10 métriques (ou KPIs) représentatives du domaine des centres de calcul (Table 2.1).

Total Energy Consumption E_{DC} Elle représente la consommation énergétique (souvent à 100% électrique) d'un centre de calcul. Cette valeur historique est toujours très utilisée car elle permet de facilement suivre l'évolution d'un centre de calcul, mais aussi d'en dimensionner la partie électrique. Par contre elle est inopérante pour en comparer deux du fait des différences d'usages et de matériels.

Power Usage Effectiveness PUE Une métrique plus complexe est le PUE[3]. De manière générale il s'agit du ratio entre l'électricité utilisée en tout dans le centre de calcul et celle utilisée pour la partie informatique. Mais dans ce cas là, doit-on compter les ventilateurs des cartes mères dans cette dernière ? C'est la raison de l'existence de plusieurs niveaux de PUE, de PUE_0 à PUE_4 en fonction du périmètre exact de la partie informatique. De manière générale on a donc $PUE = E_{DC}/E_{IT}$. Pour le PUE,

1. `http://www.futurefacilities.com/`

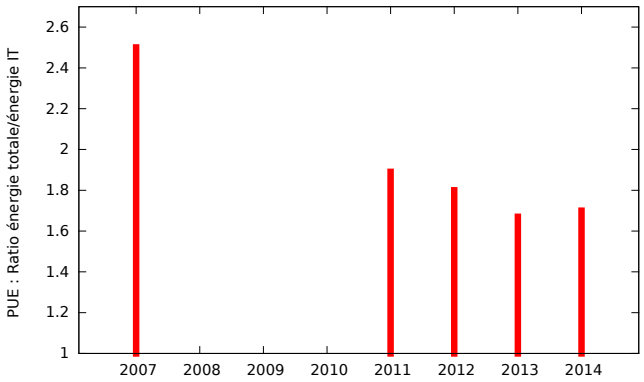


FIGURE 2.9 – Évolution du PUE de 2007 à 2014, étude de l'*UptimeInstitute*

E_{IT} représente la consommation des serveurs, ventilateurs, stabilisateurs de tensions, batteries, mais pas des groupes froids associés au refroidissement. Pour le PUE_3, E_{IT} est la partie consommée par les serveurs et routeurs en eux même.

Le PUE peut par exemple servir lors d'une mise à jour d'un système de climatisation. Les différentes offres pourront par exemple fournir leur PUE dans leur réponse d'appel d'offre. Ces valeurs, en plus d'autres telles que le prix permettront alors d'aider à la prise de décision. L'idée du PUE est que si il baisse, alors l'efficacité augmente. Dans les faits, ces dernières années le PUE a globalement diminué[95] (Figure 2.9) et dans le même temps, l'efficacité énergétique a augmenté. Par contre il est difficile d'évaluer précisément cette augmentation en n'utilisant que la valeur du PUE.

Afin de rendre plus précis le calcul du PUE plusieurs améliorations ont été apportées en mettant une limite plus précise au travail efficace. Le PUE_4 considère ainsi que le travail efficace est uniquement celui utilisé sur la carte mère, sans compter les ventilateurs ni l'alimentation ni toutes les autres fonctions support considérées comme inutiles.

Pourtant comme le montre la figure 2.10 la part utile est encore plus petite. Cette figure montre que la partie utile est encore bien plus faible car une partie de la puissance est perdue dans des courants de fuite à cause de l'augmentation de la température du processeur.

Mais pour aller plus loin, il faudrait revenir en 1961[61] quand Rolf Landauer d'IBM a étudié grâce aux principes de la théorie de l'information la limite théorique de l'énergie nécessaire pour manipuler de l'information. On arrive alors à des systèmes qui théoriquement n'utilisent pas d'énergie si l'entropie ne change pas, i.e. si on ne perd pas d'information dans le traitement des données[8].

Energy Wasted Ratio EWR Cette métrique est obtenue grâce à : $E_{DC}^{Useless}/E_{DC}$ et donne des informations sur les gains potentiels au niveau de la gestion des machines et des logiciels. $E_{DC}^{Useless}$ mesure la consommation liée aux serveurs inutilisés ou en cours de maintenance. Cette valeur, analysée en duo avec E_{DC} permet d'avoir une vision fine des gains possibles et permet de proposer des leviers pour améliorer un centre de calcul.

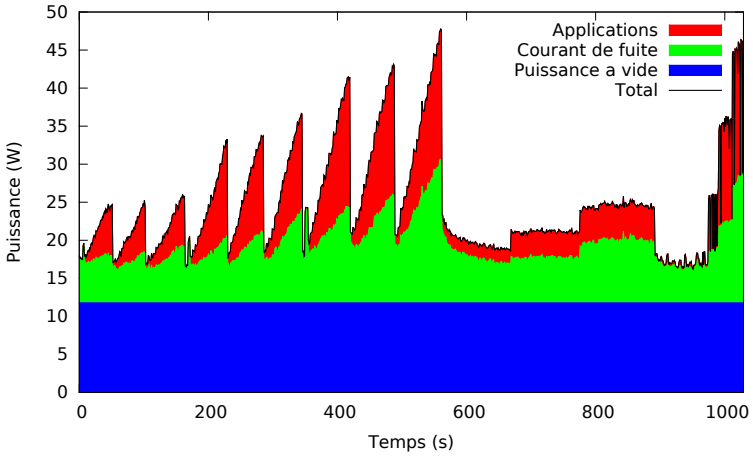


FIGURE 2.10 – Consommation d'un serveur. En bas est représentée la partie fixe. La seconde partie (en vert) vient des pertes de courant du fait de la température du processeur. Enfin la partie rouge représente la puissance consommée par l'application.

Productivity *Prod* La productivité est une sorte de Saint Graal[105] de l'évaluation des centres de calcul. De manière générale la formule est la suivante : W_{useful}/E_{DC}. Le problème vient du fait qu'il est impossible de comparer le travail efficace (W_{useful}) entre plusieurs applications différentes. Cette valeur n'est possible que dans le cas de systèmes exécutant des applications homogènes comme des serveurs webs (en requêtes/s) ou du calcul scientifique (en FLOPS ou *FLoating-point Operations Per Second*). Cette valeur est alors très pertinente lorsqu'on veut comparer deux centres de calcul ou même deux applications appartenant exactement au même domaine.

Rack Cooling Index $RCI_{LO,HI}$ Il s'agit d'une métrique souvent obtenue par simulation de type CFD (*Computational fluid dynamics*) qui évalue la qualité du refroidissement des serveurs. RCI_{LO} l'évalue par rapport à la température minimume recommandée, RCI_{HI} fait de même par rapport à celle maximale. Cette métrique permet d'obtenir une information synthétique d'une simulation de la répartition de la température dans un centre de calcul. En effet ces simulations fournissent une grande masse de données qu'il peut être difficile d'exploiter. Cette métrique permet de détecter facilement l'existence d'un problème mais ne permet pas d'en détecter finement la source.

Imbalance of CPU temperature *IoT* Cette métrique permet d'évaluer le différentiel de température entre les processeurs. Les processeurs sont les principaux producteurs de chaleur dans les serveurs et sont ceux qui sont les plus variables. Elle permet de détecter sur du moyen terme des problèmes structurels de gestion du refroidissement mais aussi d'évaluer une politique de placement des applications.

Capital Expenditure $CAPEX$ Les dépenses d'investissement nécessaires à acheter et mettre à jour le matériel : bâtiment, serveurs, réseau, ...

Operation Expenditure $OPEX$ Les dépenses opérationnelles d'un centre de calcul sont principalement liées à sa consommation électrique. Des postes moins importants sont la maintenance et les licences logicielles.

Carbon Emissions CO_2 L'impact écologique peut être vu selon un point de vue local (lié à la consommation électrique), ou selon un point de vue global (tenant compte de la fabrication et du recyclage du matériel). De concert avec E_{DC}, cette métrique permet d'évaluer l'efficacité de l'utilisation d'énergies renouvelables.

2.3.2 Applications utilisées pour l'évaluation des centres de calcul

Pour finir de spécifier une métrique, un dernier point est primordial : l'application utilisée. En effet, en fonction de l'application, les mesures vont être différentes. Il est donc nécessaire de spécifier exactement l'application lorsqu'on donne un chiffre de métrique de façon à obtenir un résultat comparable.

Dans certains domaines, tel que le calcul haute performance, il existe des applications de référence pour l'évaluation. Pour le calcul haute performance (HPC), la suite *linpak* est la référence[30]. Cette suite effectue un certain nombre d'opérations matricielles qui sont considérées comme représentatives du comportement des applications de calcul haute performance.

En dehors de ce domaine, du fait de la variété des situations possibles, il n'existe pas de suite d'applications considérées comme représentatives. Chaque domaine a la sienne. Par exemple, pour les applications de type web, la suite TCP-W[1] a été pendant longtemps une référence. Du fait des changements d'habitudes, elle est de moins en moins représentative de l'utilisation du web.

Une suite d'applications a donc besoin d'être suffisamment représentative de l'impact de toutes les applications. Comme il est impossible d'utiliser une application de chaque type, un crible est nécessaire pour choisir un ensemble restreint d'applications représentatives. Il est important d'avoir une bonne couverture des comportements en utilisant un faible nombre d'applications différentes.

Nous avons proposé[C14] un ensemble d'applications avec une bonne couverture des différentes ressources (mémoire, processeur) mais aussi en fonction de leur comportement temporel. En effet pour qu'une telle suite ait une bonne couverture, elle doit avoir des impacts sur la température variés. Une application rapide mais produisant à chaque instant beaucoup de chaleur aura un rôle semblable d'un point de vue thermal à une application plus longue mais dont la production instantanée de chaleur est moindre.

A partir de 18 applications provenant de différents horizons (HPC, Cloud) nous avons évalué les profils de consommation de ressources[RT8], classifié ces profils et sélectionné un sous-ensemble ayant une bonne couverture. La table 2.2 montre les différences de profils entre ces différentes applications.

Consommation de ressource	Low	Medium	High
Réseau	EP, CpuBurn, C-ray, MemLoop, OpenSSL, PyBench, ABINIT, test3d, tar, hadoop, JBOSS	CG, SP, BT, LU	IS, FT, linpack
Mémoire	EP, CpuBurn, OpenSSL, PyBench, ABINIT	FT, SP, BT, LU, CG, C-ray, linpack	IS, MemLoop, test3d, tar, hadoop, JBOSS
Cache	MemLoop, JBOSS	CG, SP, LU, IS, PyBench, test3d, linpack, tar	FT, EP, BT, CpuBurn, OpenSSL, C-ray, ABINIT, hadoop
Branch Instruction	C-ray, hadoop, JBOSS tar	SP,BT, EP, LU, CpuBurn, OpenSSL, PyBench, test3d	IS,FT, CG, MemLoop, ABINIT, linpack
Impact			
Puissance électrique	IS, EP, Pybench, MemLoop, tar, hadoop, JBOSS	FT, C-ray, test3d, linpack	SP,BT,LU,CG, OpenSSL, CpuBurn, ABINIT
Longueur (en temps)	EP, IS, Pybench	CG, FT, ABINIT, linpack, tar	SP, BT, LU, CpuBurn, OpenSSL, C-ray, MemLoop, test3d, hadoop, JBOSS
Impact thermique	EP, IS, Pybench, linpack, tar, hadoop, JBOSS	FT, CG, MemLoop, test3d	LU, SP, BT, CpuBurn, C-ray, OpenSSL, ABINIT

TABLE 2.2 – Caractéristiques en consommation de ressources et en impact de 18 applications HPC (EP, IS, LU, SP, BT, FT, CG, linpack, ABINIT), synthétiques (MemLoop, CpuBurn, tar, Pybench) et Cloud (hadoop, JBOSS, test3d, C-ray, OpenSSL).

Les différentes caractéristiques intéressantes pour avoir une bonne couverture sont :

Même ressources/Puissance différente

Ressources différentes/Puissance identique

Ainsi la suite logicielle *EP, CpuBurn, IS, LoopMemory, tar, test3d et CG* permet d'avoir une bonne couverture en un faible nombre d'applications. Cette suite est donc adaptée à une évaluation de centre de calcul, en tenant compte des ressources mais aussi de l'impact énergétique et thermique. Dans d'autres cadres, il sera nécessaire d'utiliser d'autres types de suites comme nous le verrons dans la section 6.2.2 concernant l'optimisation de la consommation électrique d'un serveur. Par exemple lorsqu'on s'intéresse uniquement à la quantité d'électricité nécessaire au niveau d'un serveur, l'impact thermique étant négligé, tout comme l'infrastructure, l'utilisation de la suite proposée ici n'est pas pertinente, et l'utilisation d'applications classiques sera parfaitement suffisant.

La dernière caractéristique nécessaire pour avoir un cadre d'évaluation est de définir un scénario. Ainsi la suite de test SPECPower[62] commence par une charge maximale, puis régulièrement décrémente la charge jusqu'à 0. Ainsi tout est spécifié et reproductible. Souvent un tel scénario se veut reproduire un comportement réaliste. Ainsi dans [C6] nous avions modélisé les profils d'applications pour les grilles de calcul afin de pouvoir générer par la suite des profils réalistes pour évaluer des grilles de calcul. Pour les centres de calcul plus généralistes que ceux dirigés vers le calcul haute performance, il est compliqué de trouver des traces de systèmes réels permettant de produire un modèle. En effet, ces données sont souvent considérées comme étant un enjeu de propriété intellectuelle fort.

2.4 Limites des métriques classiques simples

L'objectif des métriques est de parvenir avec une information simple à comprendre un phénomène complexe. Historiquement il s'agissait principalement d'évaluer la vitesse de calcul globale d'un ensemble de machines identiques pour des applications clairement identifiées. Autant les objectifs que les méthodes étaient clairs.

L'un des grands avantages de l'apparition des centres de calcul reposant sur la virtualisation a été la segmentation des métiers et donc une meilleure spécialisation des acteurs. Un opérateur de centre de calcul se concentre sur sa gestion sans avoir à se poser de question sur les applications en jeu. Réciproquement, les utilisateurs n'ont pas besoin d'avoir d'informations particulières pour pouvoir utiliser indistinctement n'importe quel centre de calcul. Le revers de la médaille vient de la multiplicité d'objectifs résultant de ce découpage. De plus les centres de calcul sont devenus beaucoup plus grands et complexes. Leur complexité vient de leur taille, mais aussi découle aussi de l'hétérogénéité de leur matériel et des applications qu'ils exécutent.

Une dimension supplémentaire vient de leur intégration dans les villes. Leur consommation ou production d'électricité, de chaleur, mais aussi leur taille fait qu'ils ne peuvent pas être considérés dans l'absolu comme un objet éthéré mais doivent maintenant être pensés par et pour leur intégration dans les villes, mais aussi au niveau développement du territoire.

Toutes ces contraintes et la multiplicité des acteurs font que les métriques actuelles ne sont plus pertinentes.

Le PUE qui est le chiffre mis en avant lorsqu'une publicité est faite autour d'un centre de calcul est totalement insuffisante pour le décrire.

2.4.1 Limites du PUE

Un exemple simple permet de comprendre la limite du PUE. Prenons deux centres de calcul : A et B. Supposons que A et B consomment autant pour la partie IT (de 200 à 400 suivant la charge), mais que A consomme toujours 100 (référence) pour le refroidissement quelque soit la charge, et que B consomme de 0 à 120 en fonction de la charge.

A pleine charge, A sera considéré comme plus efficace (PUE=1.25) que B (PUE=1.3). Pourtant en fonction des applications A et B seront suivant les cas plus ou moins intéressant à choisir. À moyenne charge par exemple, A consommera 400 (PUE=1.33) contre 360 pour B (PUE=1.2).

Pour aller plus loin, si l'opérateur de A veut améliorer artificiellement son PUE, il lui suffit de dégrader la qualité des applications exécutées afin que la charge IT soit le plus souvent le plus proche possible de 100%. Ainsi son PUE se réduira et donc s'améliorera en même temps que ses applications perdront en efficacité et que pour le même travail effectif sa consommation d'énergie augmentera.

Contrairement à leur usage, le PUE et les autres métriques décrites plus haut ne suffisent pas à tous les usages. Ainsi le concepteur d'un centre de calcul voulant dimensionner la climatisation sera intéressé par l'IoT (*Imbalance of CPU temperature*) qui lui permet d'utiliser efficacement son infrastructure de refroidissement. Un opérateur sera plus intéressé par l'$OPEX$ qui lui permettra de gérer ses tarifs pour ses utilisateurs ainsi que ses contrats avec ses fournisseurs d'électricité.

Même ainsi un certain nombre d'éléments sont impossibles à quantifier du fait de la taille et de l'hétérogénéité des infrastructures. Au delà des infrastructures de calcul, il faut tenir compte de l'environnement, tel que la source de l'électricité (nucléaire, solaire,...), la récupération de la chaleur.

La gestion fine des métriques est donc primordiale et il est illusoire de considérer qu'une seule métrique puisse condenser toutes les informations intéressantes. Dans le cadre de CoolEmALl, nous avons ainsi dénombré dans le livrable D5.6[RT8] 32 métriques différentes concernant les centres de calcul, toutes ayant un objectif et un usage particulier.

La définition d'une ou plusieurs métriques pour but de comparaison est inutile sans les points suivants :

— **Un faible nombre de degrés de liberté**. Il est nécessaire que seul l'élément que l'on veut tester change. Ainsi pour comparer en utilisant des métriques, il faut particulièrement faire attention à la définition précise de ce qui est à comparer.

— **Une notion de référentiel d'utilisation**. Ainsi si on veut définir des ordonnanceurs, il faudra définir une base commune d'applications à ordonnancer. Dans

le cas d'une comparaison entre deux systèmes de refroidissement, la disposition de la pièce devra être définie.

— **Une unité de chaque métrique**. Une métrique doit viser un objectif particulier, et posséder sa propre cohérence. Pour optimiser à la fois la performance et l'énergie on utilisera deux métriques plutôt que de tenter d'en fabriquer une seule qui perdra son sens et risquera d'avoir des failles.

2.5 De l'utilisation des métriques

2.5.1 Différent publics

Une grande difficulté d'utilisation de ces métriques vient de la multiplicité des publics visés. En effet, la lecture des métriques est différente en fonction des intérêts de leurs lecteurs.

Associés aux métriques présentées en 2.3.1 nous avons dans le cadre du projet CoolEmAll défini les publics suivants :

Constructeur/Concepteur : Il a besoin d'outils lui permettant de concevoir la position des murs, de l'air conditionné, des racks mais aussi de dimensionner et de choisir le type des équipements de refroidissement.

Propriétaire : Ce dernier a besoin de connaître les flux financiers sur du moyen et long terme, mais aussi de fluides en général (eau, électricité, chaleur) de préférence aussi de les réduire.

Opérateur : L'opérateur veut connaître l'état instantané du centre de calcul, mais aussi avoir des systèmes d'alertes. Il s'intéresse autant aux valeurs sur lesquelles il peut influer qu'à celles qui lui permettent de détecter les anomalies.

Utilisateur : Il est principalement intéressé par des métriques de coûts et de performance à faible granularité (au niveau de ses opérations, pas à celle du centre du calcul).

Chercheur : Le chercheur quand à lui est souvent intéressé par un peu tout et plus encore. Il cherche surtout des métriques permettant de comparer afin d'évaluer des algorithmes de placement de rack, de commande d'air conditionné, ou même d'ordonnancement de machines virtuelles.

Les attentes de ces différents acteurs sont très différentes concernant les métriques. Ainsi un constructeur de centre de calcul sera intéressé le plus souvent par le comportement de ce dernier à pleine capacité. Cela lui permettra de dimensionner la climatisation, la fourniture d'électricité. L'opérateur quand à lui sera plus intéressé par la dynamique du centre de calcul, ainsi que les moyens d'actions qu'il possède pour améliorer la qualité de service fournie aux utilisateurs ainsi que maximiser ses gains.

2.5.2 Des métriques à l'objectif

Ainsi les métriques ne sont pas une finalité mais seulement une manière de définir des objectifs.

Gérer un objectif est beaucoup plus complexe que de simplement optimiser une métrique. Si on veut optimiser l'énergie consommée dans un centre de calcul, la meilleure méthode consiste simplement à tout éteindre. Dans ce genre de cas il est sans intérêt d'optimiser l'énergie sans tenir compte du tout de la performance (ou de la qualité de service).

De manière générale il existe deux méthodes pour gérer les multi-objectifs :

— On peut définir un certain nombre de seuils pour toutes les métriques qui nous intéressent, sauf pour une qu'on va optimiser sous contrainte des autres. Par exemple, on peut avoir un budget de puissance électrique maximal du fait d'un contrat avec un fournisseur d'électricité et vouloir optimiser la puissance de calcul avec cette contrainte de puissance.

— Une autre méthode consiste à faire une combinaison (souvent linéaire) de plusieurs contraintes et d'optimiser cette contrainte. Un grand problème de cette méthode vient de la nécessité de normalisation des différentes métriques pour qu'elles puissent être comparable. Dans le cas précédent on pourra par exemple utiliser, après normalisation de la puissance de calcul et électrique, $Puissance^{norm}_{calcul} - Puissance^{norm}_{electrique}$ et maximiser cette valeur.

Ces deux méthodes sont parfois utilisées conjointement lorsqu'on a un grand nombre de métriques qui sont intéressantes. On va ainsi fixer un certain nombre de seuils sur plusieurs métriques puis on va optimiser une combinaison de certaines autres métriques.

Le problème de l'optimisation d'une combinaison linéaire de plusieurs métriques est qu'il y a une forte probabilité d'avoir des grandeurs différentes qu'il est alors nécessaire de normaliser. La normalisation d'une grandeur ne consiste pas seulement à la ramener entre 0 et 1 mais aussi à assurer que sa croissance est régulière. Lorsqu'on a plusieurs métriques de différentes natures, une méthode consiste à optimiser une des métriques, puis à relâcher cette valeur pour ensuite optimiser une autre métrique. On réussit ainsi à optimiser l'ensemble des métriques sans être au détriment d'aucune autre. Nous avons utilisé cette méthode avec succès pour des centres de calcul[J9]. Une analyse plus détaillée de la notion d'optimisation multi-objective sera abordée dans la section 5.3.2.

2.5.3 Les métriques pour les opérateurs de centres de calcul

Dans le cadre des centres de calcul, comme expliqué précédemment, l'un des publics les plus intéressants est l'opérateur. En effet il existe déjà un grand nombre de centres de calcul construits et permettre à leurs opérateurs d'avoir une vue plus précise peut avoir un impact important. Ce cadre a donc un très fort potentiel d'applicabilité et d'impact.

Dans la suite le point de vue sera celui d'un opérateur de centre de calcul, ayant à sa disposition un certain nombre de leviers d'actions. Les métriques pertinentes pour l'opérateur seront celles lui permettant d'évaluer l'utilisation des leviers et de l'aider ainsi dans son choix.

Du point de vue de l'opérateur, la notion de performance est floue. En effet, pour lui, les machines virtuelles sont des boîtes noires auxquelles il n'a pas accès. Il utilise deux types de métriques. Les métriques internes (Voir Section 2.3.1) qui lui permettent

Catégorie	Nom	Notation classique	Fonctionnel
PERFORMANCE	Execution Time	T_{ex}	non
	Latency	α	non
	Throughput	β	non
	Response Time	T_{rep}	non
DEPENDABILITY	Accuracy	Ac	non
	Reliability	Re	non
	Availability	A	non
	Capacity	$C_{Cluster}$	non
	Scalability	S	non
	Reactivity	Ra	non
	Dynamism	Dyn	non
	Robustness	Rob	non
	Stability	St	non
	Fault Tolerance	FTs	non
	Sustainability	$Sust$	non
	Agility	$\oslash$	oui
	Insurance	$\oslash$	oui
	Usability	$\oslash$	oui
	Customization	$\oslash$	oui
	Automatic Update	$\oslash$	oui
SECURITY	Authentication	$\oslash$	oui
	Authorization	$\oslash$	oui
	Integrity	$\oslash$	oui
	Confidentiality	$\oslash$	oui
	Accountability	$\oslash$	oui
	Traceability	$\oslash$	oui
	Encryption	$\oslash$	oui
	Data Life Cycle	$\oslash$	oui
	Non-Repudiation	$\oslash$	oui
COST	Service Cost	ω	non
	Energy Cost	E	non
	Carbon Cost	CFP	non

TABLE 2.3 – Tableau récapitulatif des notations des paramètres de QoS

d'évaluer le comportement de son infrastructure. Les métriques externes, sur lesquelles il s'engage avec ses clients.

La qualité de service[103] sur laquelle il s'engage avec l'utilisateur est sur de la mise à disposition de ressources. De manière générale, l'opérateur et l'utilisateur s'engagent sur un SLA (*Service Level Agreement*) utilisant un certain nombre de métriques de QoS (*Quality of Service*) sur lesquelles l'opérateur s'engage.

Il existe deux types de QoS : fonctionnelle (le système est-il crypté) ou non fonctionnelle (quelle capacité mémoire). La table 2.3 montre les éléments de QoS existant et étant à la base de la plupart des SLAs commerciaux actuels. Une définition précise de chacun de ces éléments peut se trouver dans [J8] où nous décrivons les métriques, mais aussi des cadres de leur utilisation.

L'objectif de l'opérateur est donc de réussir à optimiser des métriques internes tout en atteignant la qualité de service vendue aux utilisateurs.

2.6 Conclusion

Modéliser un centre de calcul est complexe et ne peut être réalisé qu'avec un but. En effet, il est possible de complexifier la modélisation sans fin. La question primordiale sera donc : *Pour quel usage ?*

Une fois répondu cette question, la difficulté viendra du dosage. En effet, modéliser très finement la gestion en fréquence du processeur d'un point de vue énergétique et thermique est peu pertinent si on le combine avec un modèle simpliste d'application ne dépendant pas de la ressource allouée.

Pour une expérience ou étude donnée, il sera alors important de prendre un front de profondeur cohérente entre les modèles. Il serait intéressant de faire une classification des modèles globaux obtenus en fonction de la précision des sous-modèles. Cette classification pourrait donner des informations sur le type de métriques ayant un sens pour chaque ensemble de sous-modèles.

Pour l'ensemble de ces modèles, l'interdépendance est un point commun. Le modèle du matériel interagit avec celui des applications. Ces deux modèles interagissent avec la boucle thermique. Une simplification souvent utilisée consiste à étudier alors les états stables. Mais nous avons vu qu'il est aussi possible d'étudier les états de transitions et que ces derniers sont importants. La montée en température du matériel ou la migration d'une machine virtuelle ne sont pas instantanées.

Le problème de l'optimisation de l'économie d'énergie est qu'il dépend de beaucoup de phénomènes : comportement des applications, du matériel, du système de refroidissement, mais aussi objectifs des opérateurs et des utilisateurs.

On se place donc dans le cadre des opérateurs de cendre de calcul. Ces derniers veulent optimiser autant leur OPEX que leur qualité de service afin de garder leurs clients. Certains pour d'autres raisons que financières ont aussi comme objectif de réduire leur empreinte écologique. Il est à remarquer qu'il existe une profusion de métriques différentes. Certaines de ces métriques tentent d'agréger plusieurs métriques en une seule pour donner une sorte de label à un centre de calcul, comme pour les frigidaires. Pourtant,

comme nous l'avons vu avec le PUE, cette agrégation conduit à une perte d'information. Il n'est possible d'évaluer la qualité d'un centre de calcul qu'en utilisant un bouquet de métriques du fait de la complexité du système étudié. De plus, la question du scénario lié à la mesure de la métrique reste ouverte. Chaque centre de calcul utilise son environnement de production pour évaluer les métriques, ce qui rend les comparaisons complexes voire impossibles.

Pourtant les outils décrits ici peuvent être utilisés à d'autres fins. Par exemple, dans le projet CoolEmAll, nous avons étudié[J9, C17] l'impact de la position des serveurs dans un centre de calcul en tenant compte des aspects thermiques. Ainsi ces travaux principalement destinés aux opérateurs de centre de calcul existants peuvent être utilisés par d'autres communautés tels que les concepteurs de centres de calcul.

Certains modèles mériteraient d'être améliorés, principalement au niveau système qui est un niveau moins étudié récemment. Ainsi la problématique de traduction de profil sur d'autres serveurs est encore à valider dans un cadre plus général. La problématique du comportement d'un point de vue ressources de l'exécution de plusieurs machines virtuelles est connu expérimentalement mais peu de modèles généraux la décrivent.

Enfin de manière générale un modèle est d'autant plus utile qu'il est facilement instantiable. Si il est nécessaire d'avoir des informations trop complexes à obtenir, le plus beau modèle ne sera pas utilisé. C'est pour cette raison que des outils d'instantiation automatique de modèles (tels que hwloc[16]) sont très intéressant. C'est aussi la raison pour laquelle les modèles thermique ou de puissance décrits dans ce chapitre utilisent des valeurs telles que P_{min}, P_{max}, T_{min} et T_{max} qui sont facile à mesurer et non pas des constantes de type *capacité thermique*.

Chapitre 3

Connaissance et degrés de liberté

> Thou must count to three.
> Three shall be the number of the counting and the number of the counting shall be three.
> Four shalt thou not count, neither shalt thou count two, excepting that thou then proceedeth to three.
> Five is right out.
>
> Quest for the Holy Grail
> Monty Python

Le chapitre précédent présente les travaux de modélisation de centres de calcul. Dans un centre de calcul réel, le nombre de degrés de liberté est limité. On aimerait pouvoir utiliser un modèle global idéal tel que celui présenté en Figure 3.1.

Cette figure représente un centre de calcul dont les besoins en électricité suivent fidèlement la charge. Dans ce cas le choix de la machine sur laquelle s'exécute une tâche n'a pas d'importance. Il n'y a pas besoin d'avoir beaucoup d'informations ni d'utiliser de leviers car toutes les décisions ont un impact équivalent.

Pourtant dans la réalité on retrouve plutôt le second comportement montré sur cette figure. On va devoir gérer des coûts temporels et énergétique d'allumage et d'extinction,

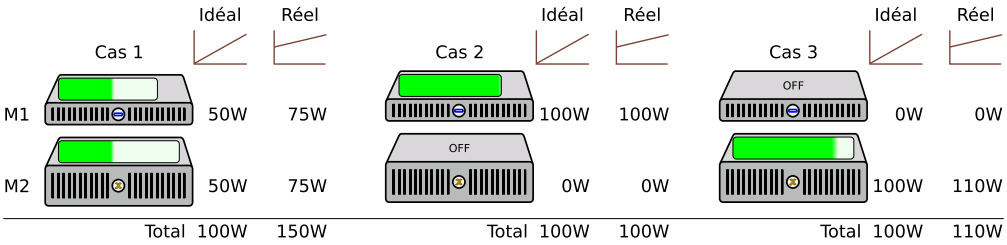


FIGURE 3.1 – Exemples de comportements de centres de calcul, idéal proportionnel et réel. M1 (resp. M2) consomme de 0 à 100 W (resp 0 à 120W) en proportionnel, et 50 à 100W (resp 60 à 120W) en réel.

mais aussi des coûts statiques qui représentent une grosse déperdition par rapport à un système idéal. Une des raisons vient du caractère fixe et discret du système. Ainsi un serveur est soit allumé soit éteint, a un nombre de modes fini. Par mode on parle ici de l'ensemble des états de ses composants. Les processeurs, mémoires, cartes réseaux, disques ayant des modes de fonctionnement discrets et finis, il en est de même pour les serveurs eux même. Ce raisonnement s'étend aux autres éléments tels que l'infrastructure de réseau ou de refroidissement.

Dans ce cadre réel, quelles sont donc les méthodes pour améliorer la qualité d'un centre de calcul. Le chapitre précédent montre comment évaluer de manière raisonnable cette amélioration.

Pour améliorer la qualité d'un centre de calcul il est donc nécessaire de recenser les différents leviers disponibles. Les leviers les plus intéressants sont ceux qui peuvent être utilisés dans les centres de calcul sans modification physique, et donc peuvent être utilisés de manière logicielle.

- Il existe différent objets pour ces leviers
  - Local/physique : Utilisation du DVFS, allumage ou extinction total ou partiel des serveurs, routeurs,...
  - Global/logiciel : Gestion des logiciels, de leur emplacement, de leur configuration, de leur déplacement ou reconfiguration.

Mais au delà des moyens d'action il y a deux autres éléments qui sont nécessaires. Avoir une vue précise du système, et être capable d'évaluer l'impact de ces leviers sur les métriques.

L'impact des leviers peut être évalué grâce à une modélisation précise du système, qui a été décrite dans le chapitre 2. On va avoir un grand nombre de leviers qui vont impacter les métriques. A un instant donné il sera donc possible de faire plusieurs combinaisons d'actions. Comme l'objectif est d'optimiser plusieurs métriques en même temps, il sera donc nécessaire d'avoir un processus d'optimisation multi-objectifs, par exemple par front de Pareto, ou par des méthodes floues. Les méthodologies d'optimisation multi-objectifs seront décrites en section 5.3.2.

Les capteurs quand à eux sont de plusieurs types. Ils peuvent être simples, tels que des capteurs de charge processeurs ou complexes, tels que des capteurs capables de tracer les trafics réseaux entre les différentes applications. On peut aussi avoir des capteurs qui servent de déclencheurs à la prise de décision tels que des indicateurs de surchauffe ou de simple évènements temporels réguliers.

3.1 Moniteur de supervision

On considère habituellement que la surveillance d'un centre de calcul est un problème déjà résolu. Plusieurs services classiques sont souvent utilisés. On retrouve dans cette catégorie Ganglia[68], Nagios[5], ou plus récemment Zabbix[104]. Les grands objectifs de ces systèmes sont d'être capables de s'exécuter dans une grande variété de cas, mais aussi de peu perturber les serveurs observés et de réagir rapidement.

Il existe pourtant deux limitations à ces systèmes. Ils se reposent sur les capteurs

présents sur les différents serveurs qui sont eux-mêmes limités, et le paradigme totalement centralisé sur lesquels ils se reposent ne passe pas à l'échelle.

Les capteurs existants sont très précis quant à l'utilisation des ressources matérielles (processeur, mémoire, réseau). Par contre lorsqu'on est en système partagé, il devient nécessaire d'avoir des informations au niveau de chaque application, d'être capable de les identifier et de faire leur profil de consommation de ressources.

3.1.1 Identification d'application

Il existe une grande variété d'applications. Dans le cadre du calcul haute performance une tendance est de considérer que les applications sont relativement simples, consommant principalement du temps processeur. Pourtant la situation est plus complexe comme nous l'avons montré lors de la description des suites logiciels de tests en Section 2.3.2. Dans un centre de calcul, le nombre différent d'applications pendant une période donnée est relativement faible. D'un point de vue échelle de temps, ces tâches sont souvent très longues par rapport aux temps de reconfiguration qui lui est de l'ordre de la minute.

Dans le domaine du cloud, la situation est assez différente. Les applications ont un comportement beaucoup plus dynamique, et un grand nombre d'applications peuvent cohabiter à un instant donné.

Comme décrit en Section 2.2 une application est décrite comme une suite de phases. Chaque phase décrit un temps où la consommation de ressources est presque constante.

Détection de phase D'un point de vue capteur, le plus simple est celui capable de produire les profils tels que visible sur la figure 2.5. Une possibilité de produire un tel profil est de considérer qu'à chaque instant de mesure, on stocke l'ensemble des mesures (appelés EV pour *Execution Vector* dans la suite) des capteurs liés à cette application. On peut alors avoir un vecteur comportant les informations mémoire, processeur, réseau,... Un profil serait alors composé de l'ensemble de ces mesures. Dans [J10], nous proposons de comparer les EVs les uns après les autres afin de découvrir les phases. Si un EV est proche du premier EV de la phase, on considère que la phase continue. Sinon, on déclenche une nouvelle phase. Le capteur de détection de phase est un capteur aussi important que celui concernant la gestion des ressources. En effet il n'est pas possible de prendre une décision chaque seconde, ce qui est le temps habituel de mesure des données. Par contre, prendre une décision à chaque changement de phase est plus raisonnable.

Identification de phase Au delà de la détection du changement de phase, il est intéressant de savoir quelle phase est en train de s'exécuter. En effet, si on exécute régulièrement une même phase, il peut être intéressant d'y attacher des recettes de gestion efficace par exemple. Ou même si une phase d'utilisation d'une bibliothèque à licence est détecté, il est possible de rajouter cette utilisation dans la facture. Enfin, si une phase est détectée très régulièrement, des ajustements adaptés peuvent être réalisés.

Cette identification est rendue difficile par le fait que pour l'opérateur de centre de calcul, les applications sont des boites noires.

Identification dans un corpus Une méthode[C13] pour identifier une phase parmi un corpus connu consiste à utiliser des outils statistiques à fenêtres sur les EVs : Il s'agit lors du début d'une phase de prendre un nombre réduit de valeurs, d'utiliser la moyenne, médiane, l'écart type, les premiers et derniers déciles et d'avoir ainsi une signature de la phase. Dans le cas d'un corpus connu relativement limité en taille, une telle signature est suffisante pour identifier une phase d'exécution parmi celle connues. En utilisant cette méthode nous avons obtenu[C13] un taux de reconnaissance de 100% pour pour un corpus composé de 7 programmes de la suite de tests NPB[28].

Identification aux phases passées Lorsqu'un corpus n'est pas connu à l'avance, il devient nécessaire de procéder en deux temps : 1) Détecter une nouvelle phase, 2) Déterminer si cette phase commençant est une nouvelle phase ou si elle a déjà été détectée. Nous avons explorés[C22] différentes techniques de reconnaissances de phases où chaque phase est déterminée par un EV caractéristique, avec un taux de réussite proche de 100%. La Figure 3.2 montre que pour une application cyclique telle que WRF les profils de consommation de ressources (et donc les EVs) des différentes phases sont très semblables. En effet, chaque point y représente la similitude de consommation de ressource entre deux instants. Un point noir indique que les deux consommations ont été identiques. La couleur de chaque point est obtenu en faisant la distance entre les EV à ces deux instants. Un triangle noir au contact de la diagonale indique une phase où la consommation de ressource est constante. Un carré noir indique que deux telles phases sont proches d'un point de vue de leur consommation de ressources. On peut donc retrouver grâce aux valeurs mesurées le fait que WRF est une application cyclique.

Caractérisation d'une phase Au delà de reconnaître une phase déjà passée, il est important de reconnaître les caractéristiques principales d'une phase. En utilisant les EVs et en se basant sur des suites logicielles connues, nous avons par exemple montré[C23] qu'il est possible de dire qu'une phase est dans la liste suivante : *Compute intensive, Memory intensive, Communication intensive, IO intensive.* Il s'agit de déterminer le goulet d'étranglement de la phase en fonction de son profil de consommation de ressources.

Détection de phase externe Un élément commun de ces méthodes est leur coté intrusif au niveau système. Même si les algorithmes de comparaison utilisés sont simples, il reste nécessaire d'exécuter le détecteur de phases ce qui peut avoir un impact sur l'application elle-même en la ralentissant. De manière semblable aux méthodes de détection du type d'activité dans une maison basés uniquement sur un wattmètre intelligent[90], nous avons développé une méthode[C13] utilisant uniquement des mesures externes (puissance électrique et trafic réseau) pour la composition de l'EV.

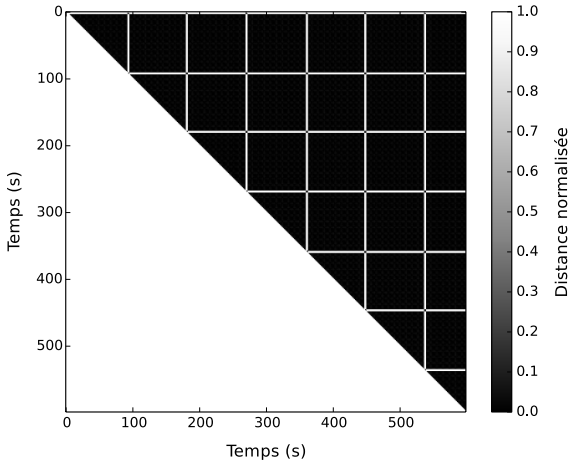


FIGURE 3.2 – Matrice de distances entre les comportements d'une application (WRF, *Weather Research and Forecasting*). Chaque point montre la similarité dans la consommation de ressources à deux instants différents. Les lignes verticales montrent la similarité de consommation de ressources entre un instant (abscisse) et le passé.

3.1.2 Capteurs systèmes sur informations indirectes

Autant il est relativement simple d'obtenir le profil de consommation pour les ressources simples (processeur, mémoire), autant il devient difficile de le faire pour des ressources un peu plus fine du fait des limites des systèmes d'exploitation actuels. Ainsi avoir des informations cohérentes sur les communications entre applications est difficile car il faut reconstruire ces informations en ayant une vue globale de quelle application possède quel port sur quelle machine à tout moment. Dans le cadre de ma participation au projet ANR SOP [1] (*think global Services for persOnal comPuter*, 2011-2014), j'ai développé un tel capteur. Ce capteur permet de lier les quantités de communications réseau aux VMs ou aux applications. Une telle information permet de migrer sur un même serveur les VMs communiquant le plus entre elles ou tout au moins d'avoir un retour sur quelle application communique avec quelle autre application.

3.1.3 Capteurs par apprentissage

Les capteurs classiques se reposent sur des valeurs systèmes et parfois des équations connues. C'est le cas du capteur associant à une application (un ensemble de processus) son trafic réseau.

Dans certains cas, la modélisation directe est trop complexe et il est alors possible d'utiliser un capteur par apprentissage. Cette notion a été évoquée en section 2.2.2 concernant la puissance d'une machine. En effet, la méthode la plus simple pour calculer

1. `http://projects.laas.fr/SOP/`

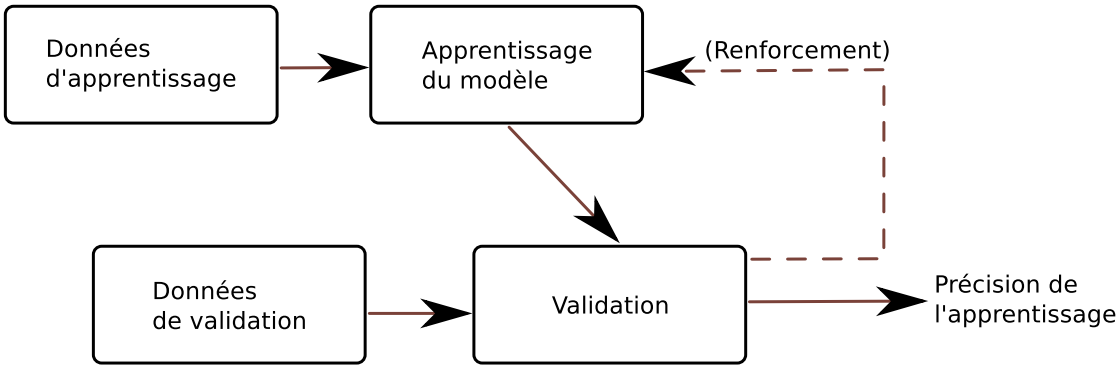


FIGURE 3.3 – Système d'apprentissage. Dans certains cas, l'apprentissage se fait en partie ou totalement grâce à une boucle de renforcement (en pointillée)

la puissance est explicitée par la formule $P_{min} + Charge(P_{max} - P_{min})$. Cette formule a le mérite de la simplicité mais sa précision est faible. Dans le cas d'un serveur Xeon, l'erreur moyenne est d'environ 15%[87]. Pour obtenir une valeur plus précise il est alors nécessaire de complexifier le modèle, mais aussi d'adapter le modèle à chaque serveur ou matériel.

Un moyen d'obtenir une précision plus grande à coût de développement moindre consiste à utiliser des systèmes à base d'apprentissage tels que montré en figure 3.3.

Le modèle alors généré le sera en fonction des données en entrée. Un avantage est qu'il n'est plus nécessaire d'avoir une modélisation dédiée. Il suffit d'avoir un ensemble de données mesurées sur un serveur qui serviront de base d'apprentissage pour produire un modèle qui lui est dédié. Les données en entrée sont les sorties des autres capteurs et peuvent être des valeurs systèmes (charge processeur, mémoire, réseau), des compteurs de performances des processeurs, ou tout autre valeur mesurable.

Dans un système d'apprentissage, les deux phases sont l'apprentissage en lui même, puis l'utilisation. La première phase est souvent très coûteuse, mais il est nécessaire que la seconde soit rapide, car elle devra s'exécuter de manière non-intrusive comme tous les autres capteurs présents sur le serveur.

Les modèles utilisés sont souvent très simples, polynomiaux ou de type réseaux de neurones par exemple. Il existe trois écueils à éviter : les biais lors de l'apprentissage, la couverture de cet apprentissage et enfin, l'impact d'utilisation du modèle.

3.1.3.1 Les biais lors de l'apprentissage

Les biais sont de différents types. Ainsi il est nécessaire de prendre en compte le contexte. Pour une mesure de puissance, il faut tenir compte de la courbe d'efficacité de l'alimentation comme vu en section 2.1.2.1.

Un autre élément important concerne la précision du matériel de mesure. En effet beaucoup de sites de Grid'5000 ont des wattmètres précis à 2 ou $3W$ par exemple. Une

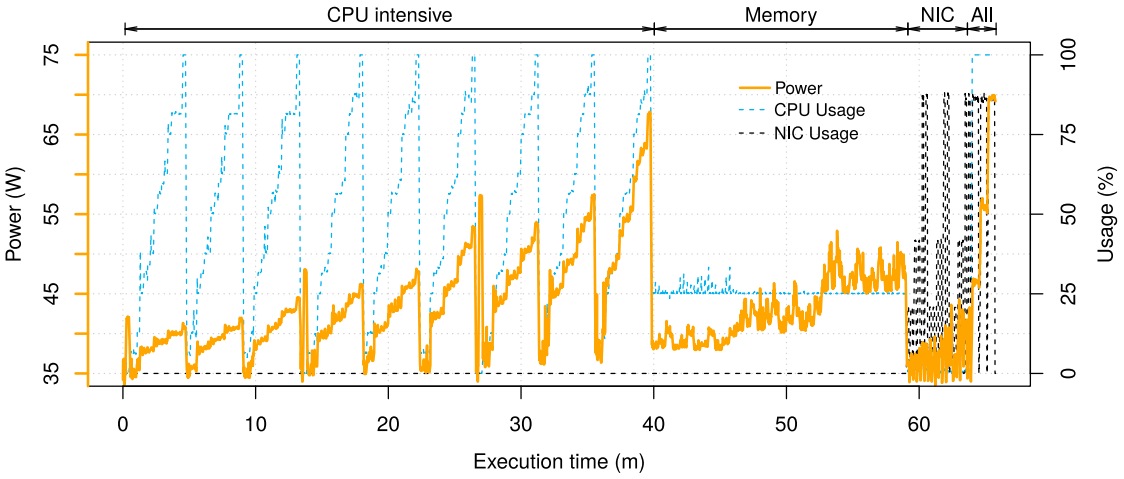


FIGURE 3.4 – Charge synthétique permettant d'avoir une couverture de tous les états du système

mesure de 10W sur ces machines devra être sujette à caution. Sur les expérimentations présentées dans ce mémoire, nous avons utilisé pour ce qui concerne les méthodes par apprentissage un wattmètre précis à $.1W$, et pour l'évaluation des heuristiques les wattmètres disponibles sur Grid'5000.

Les biais méthodologiques s'ajoutent du fait de la structure particulière de la récupération de données pour l'apprentissage. Par exemple, lorsque nous avons commencé les expériences de modélisation de la puissance d'applications[C9] nous avions une variable qui avait une grande importance : le temps. En effet, suivant l'exemple de la suite d'évaluation SPECpower[62], nous diminuions la charge avec le temps, et donc cette variable suivait assez fidèlement la puissance. Il ne s'agissait bien sûr que d'un artefact. Donc même sans pré-sélectionner complètement les variables, il faut les filtrer et ne garder que celles qui ont un sens ou tout au moins avoir un ensemble de mesures très variées.

3.1.3.2 Couverture de l'apprentissage

Ensuite il est nécessaire pour faire un apprentissage d'avoir exploré un maximum d'états possible du système. Dans [D2] nous avons utilisé cette méthode pour modéliser la puissance électrique en fonction des autres capteurs. Nous avons dans un premier temps utilisé la charge générique explicité en Figure 3.4 qui est composé de trois phases. Une première phase qui explore les différents modes du processeur (fréquences et charge), puis une seconde phase qui explore les différentes charges sur la mémoire, une phase couvrant le réseau, et enfin une phase mélangeant les différents comportements.

Il faut remarquer que la notion de couverture est différente ici de la couverture nécessaire à l'évaluation d'un centre de calcul comme décrit en Section 2.3.1. Dans le cadre de l'évaluation par métrique, il est nécessaire d'avoir une bonne couverture des

utilisations possibles. Ici il s'agit d'une couverture du domaine de valeurs que peuvent prendre les mesures de base, ie les valeurs systèmes et de compteurs de performance mesurables.

Nous avons étudié[D2] la couverture de plusieurs applications en plus du programme générique. Le Tableau 3.1 montre pour la charge générique ainsi que pour plusieurs programmes de type calcul haute performance les couvertures de domaine croisées. Chaque ligne montre le nombre de capteur (parmi 220 mesurées) dont les valeurs sont hors du domaine de charges références. Par exemple, *pybench*, a 12 variables qui ont des valeurs plus grandes ou plus petites que celles de la charge générique lorsque celle ci est utilisée en référence. D'un autre coté, la charge générique a 183 variables qui sont hors des bornes du programme *pybench*. Dans un système à apprentissage il sera donc plus intéressant de récolter un jeu de données en utilisant la charge générique que le programme *pybench*. Dans le Tableau 3.1, le programme de référence qui donne le meilleur résultat est en gras. On peut voir que même si la charge générique n'a pas une couverture totale, elle a un défaut de couverture de 39 sur 220 variables, le plus bas des différentes applications montrées ici.

Avoir une bonne couverture est donc un problème complexe. Cette couverture doit être résistante au changement de machines. Créer en fonction d'une machine une charge adaptée fournissant une bonne couverture reste un problème de recherche ouvert. Une méthode à base d'algorithmes génétiques serait intéressante à utiliser pour créer un ensemble de micro-programmes représentant une couverture maximale.

3.1.3.3 Réduire l'impact

Pour pouvoir être déployé, un capteur doit avoir une empreinte réduite. L'impact principal d'un tel système par apprentissage n'est pas tant le calcul en lui-même mais la récupération des valeurs. En effet, dans l'exemple précédent, nous récupérions 220 valeurs ce qui a un fort impact sur le système. Sur notre plate-forme, mesurer ces 220 variables une fois par seconde au niveau système (`sys`) rajoute en moyenne $.87W$ (Voir figure 3.5. Si cette opération est réalisée en ne tenant compte que des processus actifs et des valeurs qui leur sont associées, la surconsommation est alors de $.95W$ (`pid_f`), et $1.31W$ si tous les processus sont mesurés (`pid_a`). Pour référence, le système à vide consomme $30.5W$. De plus cette mesure a un fort impact sur la reproductibilité du système comme le montre la dispersion des valeurs de consommation sur la figure 3.5.

Pour un système à apprentissage il est donc pertinent de réduire le nombre de variables suivies afin de réduire l'impact lors de l'utilisation du système de mesure. La méthode classique[49] consiste à prendre l'ensemble des variables, créer un modèle pour chacune de ces variables et garder le meilleur et ainsi sélectionner une variable particulièrement intéressante. On recommence avec les variables restantes jusqu'à un seuil d'amélioration ou de nombre de variables sélectionnées.

Cette méthode est viable lorsque le processus d'apprentissage est court. Lors de nos tests[D2] avec des réseaux de neurones, du fait de la lenteur de l'apprentissage nous avons préféré utiliser une méthode de résidus. A chaque étape nous avons sélectionné la variable ayant la plus forte corrélation avec le résidu (ie la puissance moins notre

TABLE 3.1 – Nombre de variables (parmis 220) du programme de test (*Comp.*) dont le domaine sort du domaine de référence des variables mesurées sur le programme *Ref.*

Comp. \ Ref.	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20
V1-Generic	0	183	181	169	181	184	172	166	170	160	162	161	164	160	164	156	164	**154**	161	159
V2-Pybench	**12**	0	99	93	97	103	83	73	75	56	66	63	64	56	64	54	64	48	60	58
V3-OpenSSL	**11**	118	0	87	98	111	82	65	83	53	60	58	59	54	55	51	62	47	63	52
V4-C-Ray	**24**	136	131	0	141	149	119	107	99	85	86	89	88	83	96	79	92	74	87	90
V5-Gmx-Villin	15	124	111	87	0	96	58	33	53	18	19	19	16	17	18	20	21	**12**	18	17
V6-Gmx-PolyCH$_2$	18	123	104	89	103	0	55	34	53	19	21	20	19	18	22	15	22	**13**	24	15
V7-Gmx-Lzm	35	134	126	103	147	145	0	37	59	17	22	23	17	**13**	29	**13**	28	14	23	19
V8-Gmx-DPPC	44	146	144	117	171	167	155	0	71	51	45	56	42	32	49	33	48	**24**	39	38
V9-HPCC_5k	**27**	141	140	113	153	152	148	137	0	113	113	111	84	88	109	45	96	**27**	95	87
V10-HPCC_5k_dist	48	153	152	130	172	176	174	155	92	0	95	110	75	78	107	43	97	**33**	93	90
V11-HPCC_10k	46	145	146	129	170	173	167	152	87	102	0	115	79	89	122	51	100	**38**	87	87
V12-HPCC_10k_dist	50	148	147	129	169	174	169	146	89	89	81	0	59	67	106	47	94	**30**	97	88
V13-HPCC_20k	**46**	146	144	129	171	174	171	154	115	117	118	125	0	102	140	73	123	49	119	122
V14-HPCC_20k_dist	**51**	152	151	134	174	176	175	162	115	117	107	128	83	0	140	75	123	**51**	125	114
V15-NPB_A	46	148	151	122	169	169	159	145	93	88	76	83	55	59	0	40	78	**25**	70	68
V16-NPB_A_dist	**56**	164	161	145	186	186	187	179	156	165	153	163	126	129	165	0	148	65	143	140
V17-NPB_B	51	149	154	132	178	180	170	156	118	112	98	108	77	78	123	60	0	**28**	91	91
V18-NPB_B_dist	**57**	160	158	143	178	178	179	173	166	165	162	170	150	153	173	134	174	0	163	154
V19-NPB_C	55	154	153	133	179	176	174	162	121	116	113	109	85	80	132	67	103	**45**	0	100
V20-NPB_C_dist	56	154	154	126	178	175	173	160	114	113	111	114	83	89	131	67	109	**50**	101	0
Average	**39**	146	142	122	159	160	146	126	102	92	90	96	75	76	102	59	92	44	87	84
Standard deviation	16	**15**	20	22	29	28	43	51	35	47	44	46	40	42	49	36	44	32	43	43

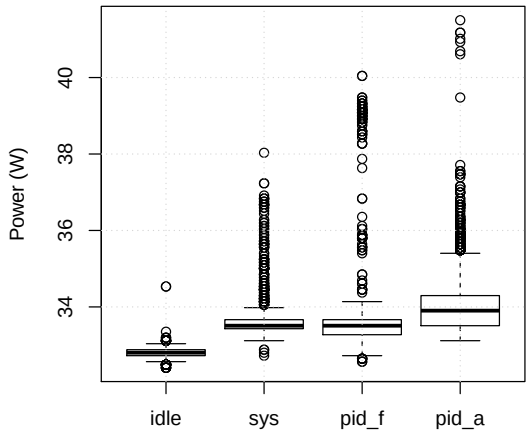


FIGURE 3.5 – Impact de récupérer des informations au niveau système (`sys`, au niveau système et processus avec et sans filtres (`pid_f`, `pid_a`) sur la consommation électrique.

modèle utilisant les variables déjà sélectionnées). Nous avons pu vérifier que dans le cas de la consommation de puissance le résultat était proche autant dans les variables (ici 8) sélectionnées que dans la qualité du réseau de neurone résultant.

De plus, réduire le nombre de variables augmente aussi le ratio échantillon/nombre de variables et permet d'avoir un meilleur apprentissage. Sur la charge générique, on arrive ainsi à passer d'une erreur moyenne de 1.33W à une erreur de 1.18W.

3.1.3.4 Un capteur de puissance de serveur par apprentissage

En utilisant les techniques d'apprentissage nous avons développé[D2] deux capteurs : Le premier (*Aggregated*) utilise un modèle additif simple basé sur la charge, la température, la fréquence, le réseau et l'utilisation mémoire ; Le second (*ANN*) utilise un algorithme à base de sélection de variables et de réseaux de neurones.

La Figure 3.6 montre la comparaison de la précision de ces deux capteurs pour lequel l'apprentissage a été fait sur le même ensemble de données appelé `learn`. Une fois le modèle appris, son erreur est évaluée pour une nouvelle exécution de chacune des applications montrées sur la figure.

Il s'agit alors de faire un choix en fonction de la précision demandée. Il est possible de faire l'apprentissage beaucoup plus rapidement en utilisant le premier capteur en réduisant drastiquement la charge d'apprentissage et en n'ayant pas besoin de la phase de sélection de variables. A l'exécution, les deux capteurs auront un coût similaire. Enfin, le second est bien plus générique et peut s'adapter à des architectures complètement différentes (ajout d'accélérateurs ou de GPU par exemple) sans modification substantielle. Il s'agit donc d'un choix à effectuer en fonction de la précision voulue, et de l'investissement associé.

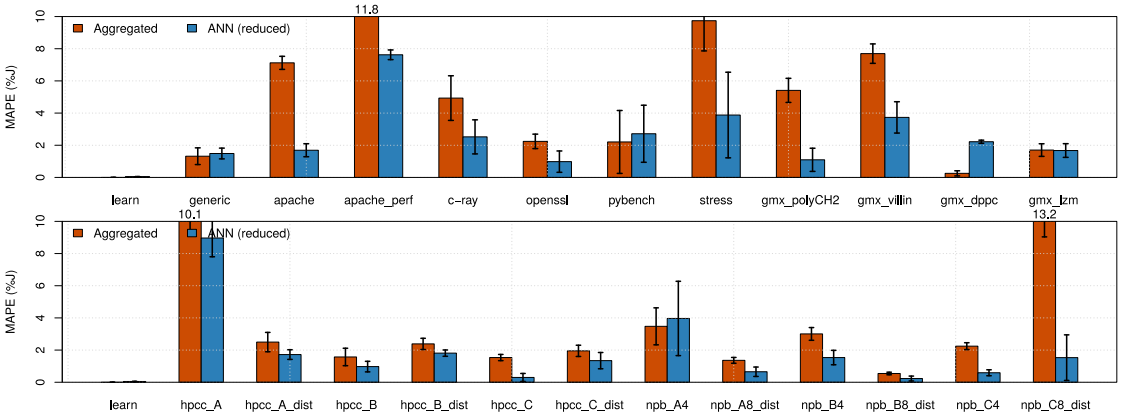


FIGURE 3.6 – Comparaison entre le modèle additif (*Aggregated*) et le réseau ne neurone (*ANN*) en utilisant comme charge d'apprentissage (`learn`) : *generic, apache, c-ray, openssl, pybench, gmx_lzm, hpcc_B_dist, npb_B8_dist.*

3.1.4 Oracles

Au delà des capteurs liés à des grandeurs physiques ou systèmes, il est aussi possible de faire des capteurs *intelligents* utilisant de telles données pour en extraire une information plus riche. Les exemples les plus classiques sont les capteurs permettant de prédire l'utilisation du système dans le futur. Il est alors possible de faire des oracles fonctionnant à plusieurs échelles. Un oracle peut simplement prédire la future consommation de ressource sans connaissance particulière[C2, 55, 46] sauf le passé récent. Ces approches utilisent la consommation de ressource récente pour prédire celle du proche futur en utilisant le plus souvent des systèmes par apprentissage : régression linéaire ou réseau de neurone. Suivant les études, l'erreur de prédiction pour du trafic de type service web est entre 20 et 40% pour ces types d'approches.

3.1.5 Le passage à l'échelle de la surveillance

La deuxième problématique est plus difficile à résoudre. En effet du fait de la taille grandissante des centres de calcul, il n'est plus possible d'avoir une vision globale du système.

Plusieurs pistes sont explorées pour tenter de résoudre ce problème. Les principales sont les systèmes hiérarchiques, les systèmes à agrégation et les systèmes statistiques.

L'élément essentiel est le lien entre le système de décision et le système de surveillance. Les deux doivent être cohérents.

Les techniques classiques de type Nagios ou Ganglia qui construisent finement les paquets de communication pour les rendre les plus légers possible ne suffisent pas pour passer à l'échelle. Lorsqu'un système fait plusieurs centaines de milliers de noeuds il n'est finalement pas nécessaire de connaître à tout instant la mémoire occupée par chaque

processus sur chaque machine.

Une direction de simplification serait d'être capable dans les algorithmes de gestion de ressource d'extraire les requêtes importantes en tant que tests de seuil et de les envoyer sur les noeuds de l'infrastructure. Il s'agit d'une sorte de généralisation des seuils de surveillance des infrastructures IPMI. On pourrait par exemple avoir des sondes intelligentes de type *latence > 10ms* ou de type *mémoire libre < 1Go*. Ce genre de conception du système de supervision a une implication forte sur la famille de systèmes de gestion de ressource possible. Une simple *consolidation* devient complexe à réaliser sans vue globale.

3.2 Leviers

Les leviers sont les moyens par lesquels le système de décision va mettre en place sa décision. Il s'agit donc de ses degrés de liberté.

Un levier est défini par :

— Son action

— L'impact de cette action sur les métriques

Un levier dont il est impossible de modéliser l'impact sur le système ne peut pas être valorisé par un quelconque système de décision. Une grande majorité des leviers agissent directement sur l'état des composants du système. Ainsi pouvoir changer les fréquences des processeurs est un levier dont l'impact fait partie du modèle de centre de calcul décrit dans le chapitre 2.

À contrario, l'impact des rayons cosmiques étant complexe à prendre en compte, il sera rare d'en tenir compte (voir Figure 3.7).

En fonction de leur granularité il existe un grand nombre de leviers différents qu'il est possible d'utiliser. Dans le cadre de l'Action COST 0804, j'ai coordonné le recensement[RT2] des principaux leviers disponibles dans un centre de calcul ainsi que leurs effets.

3.2.1 Levier au niveau d'une salle

Certains leviers au niveau d'une salle sont difficiles à utiliser de manière dynamique. Ils sont cités dans le présent document car leur importance est primordiale.

Le type de climatisation (air, liquide, avec ou sans séparation *hot/cold aisle*) est particulièrement important. Même si il ne s'agit pas d'un levier direct, il est possible d'en utiliser les caractéristiques indirectement. Lorsqu'on utilise un système de refroidissement par bain d'huile, la forte capacité thermique de l'huile peut être utilisée. Il est possible de refroidir une partie fortement utilisée en décidant d'éteindre les éléments précédent sur le chemin du fluide.

Mais ces leviers sont principalement utilisés au niveau de la conception de la salle, et sont donc de peu d'importance pour l'opérateur. Pourtant, comme nous l'avons montré lors du projet CoolEmAll, l'impact de la localisation des serveurs dans une salle ou même dans un rack est très important[C17] et peuvent avoir un impact de 3° en pleine charge

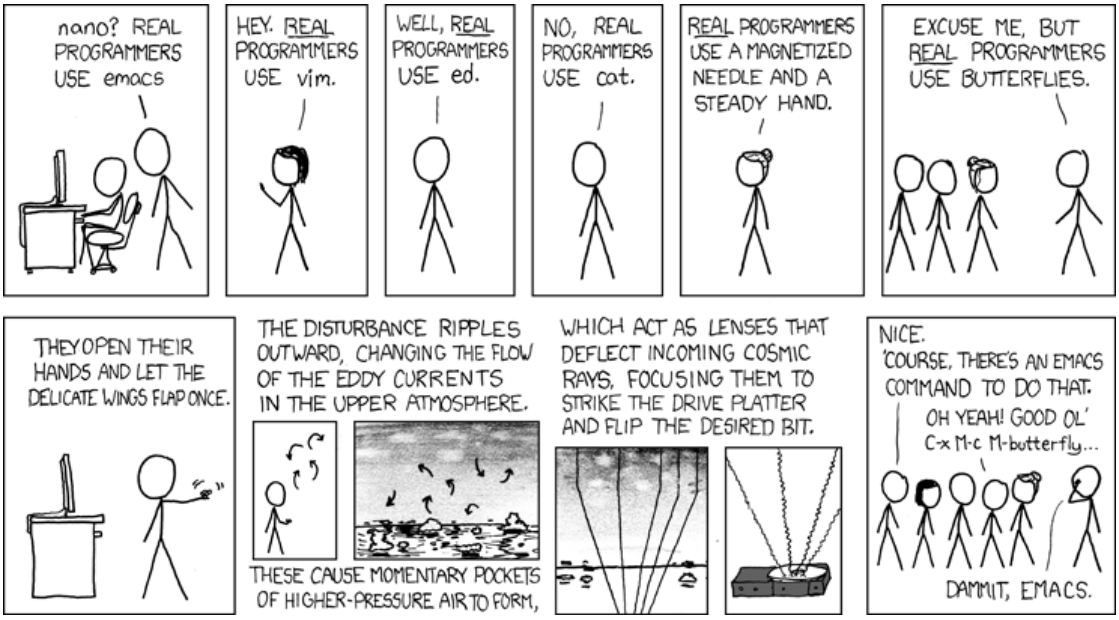


FIGURE 3.7 – license CC-ANC, Randall Munroe http ://xkcd.com/378/

sur des serveurs tournant à 30° suivant leur placement dans un bloc de serveurs haute densité.

3.2.2 Leviers matériels

On appellera levier matériel tous les leviers impactant directement le matériel informatique (processeur, mémoire, réseau,...) ou de refroidissement proche (ventilateurs, refroidissement d'armoire).

Pour le matériel informatique, il en existe deux catégories qui ne sont en fait qu'une seule : les leviers de changement de fréquence ou de capacité d'un coté, et l'extinction de serveurs ou de sous-systèmes.

Ce dernier levier n'est en fait qu'une vision extrême du premier.

Dans la partie modélisation, nous avons vu qu'il existe pour chaque sous-système un ensemble de couples *(performance, consommation)*. Par exemple, pour le processeur, ce système est appelé le DVFS (*Dynamic Voltage and Frequency Scaling*), pour le réseau il s'agit de l'ALR (*Active Link Rate*). Les niveaux d'endormissement sont associés à des performances nulles et à des consommations presque nulles. Il existe plusieurs niveaux d'endormissement qui sont caractérisés par le temps nécessaire à en sortir. Il est plus rapide de sortir d'un état *suspend-to-ram* que d'un état d'extinction total, alors que le second consommera moins.

De manière générale, pour chacun de ces sous-système il y a un coût de transition. Il faut donc associer à la liste des couples une matrice de latence de transition comme

	D'allumé à	Vers allumé	Consommation
Power-on Suspend	3s, 130W	10s, 148W	109W
Suspend-to-ram	10s, 130W	20s, 148W	7W
Suspend-to-disk	22s, 140W	55s, 150W	5.7W
Hibernate	24s, 140W	55s, 150W	2.4W
Shutdown	17s, 140W	75s, 160W	5.5W

TABLE 3.2 – Exemple de transitions entre allumé (inutilisé) et les états d'économie d'énergie pour une machine Linux équipée d'un AMD Phenom^{TM} 9500 Quad-Core[32]

l'exemple de la table 3.2 concernant les coûts pour entrer ou sortir de différents états d'économie d'énergie pour un serveur.

Une autre catégorie concerne le matériel réseau en lui même, les routeurs, commutateurs, hub. Jusqu'à récemment l'accent n'a pas été mis sur la plasticité de ces appareils et nous avons pu montrer lors de la visite pendant 3 mois d'Helmut Hlavacs en 2008 que la consommation de ces éléments était principalement lié à leur état (allumé ou éteint) et non pas à leur charge réseau[C16]. Des travaux[11] ont montrés qu'il était possible d'allumer ou d'éteindre certains de ces composants au prix d'un impact sur la redondance mais peu d'utilisation réelle de ces possibilités existent. La difficulté de mise en place, liée à la complexité de la reconfiguration des éléments de coeur de réseaux a ralenti l'adoption de ces méthodes.

En dehors de la partie extinction en elle-même, il existe plusieurs méthodes pour utiliser le levier de changement de fréquence et de capacité. Chaque sous-système peut être piloté par un système de décision globale, mais il est aussi possible de déléguer cette gestion en local sur le serveur en lui-même.

Ainsi pour les processeurs, il existe les méthodes locales suivantes :

— *Userspace*. Dans ce cas là, le serveur ne fait que recevoir des ordres de changement de fréquence. C'est le cas le plus classique lorsqu'un système de décision global contrôle la fréquence de tous les serveurs ;
— *Performance/Powersave*. Ces deux modes fixent respectivement la fréquence du processeur à la plus haute et à la plus faible. Dans les systèmes de production où le système de décision central ne gère pas la fréquence de chaque processeur, il est classique que tous ces derniers soient systématiquement à fréquence maximum ;
— *Ondemand/Conservative*. Ces deux modes s'adaptent dynamiquement à la charge. Ils essayent d'être toujours à la fréquence la plus basse permettant de traiter la charge en cours sans la ralentir.

Ces méthodes échangent l'absence de connaissance globale par une latence plus rapide.

Enfin, pour les équipements de refroidissement locaux, il y a surtout un impact sur la température et donc des effets temporels comme vu en section 2.1.2.2.

3.2.3 Leviers logiciels

Les méthodes logicielles quand à elles agissent sur les logiciels, ou sur les machines virtuelles dans le cadre du cloud.

Il existe deux types de leviers logiciels :

— Leviers externes : Démarrage, migration, suspension, sauvegarde, ralentissement ;

— Leviers internes : Checkpointing, reconfiguration.

Les leviers externes agissent sur des applications sous la forme de processus, de groupes de processus ou de machines virtuelles. Ils présentent l'avantage d'être indépendant des applications exécutées.

Démarrage Le premier levier impactant une application est celui de son démarrage. Il faut alors répondre aux questions : Où ? Quand ? Sur quel serveur ? Et à quel moment une application va être lancée ? Le choix peut être simple pour un service constant tel qu'un serveur web, ou plus complexe pour une application ponctuellement utilisée.

Migration La migration d'une application a un faible impact sur l'application elle même. La migration conserve les connexions réseaux et ne nécessite qu'une faible interruption. En 2005 déjà, dans [27] les auteurs mettent ainsi en place une migration d'un serveur web surchargé avec une interruption de service de l'ordre du dixième de seconde (165ms). Une problématique peu étudiée reste la migration entre architectures différentes (par exemple X86 et ARM). Nos résultats préliminaires[C27, J11] montrent qu'il est possible de faire migrer des machines virtuelles entre architectures complètement différentes mais au prix d'un sur-coût d'exécution sur une des deux architectures.

Suspension Un service ou un programme qui n'a pas de contrainte temporelle très forte peut être interrompu pour libérer des ressources. Pour le programme incriminé il s'agit d'une interruption invisible mais qui a un impact sur les communications. Contrairement aux migrations la connectivité réseau n'est pas assurée. La suspension conserve les allocations mémoires qui n'est donc pas libérée. La mémoire peut tout au plus être déplacée dans un moyen de stockage (disque par exemple) pour offrir aux autres applications plus de mémoire vive.

Sauvegarde Il s'agit de faire une suspension suivie du stockage de l'espace mémoire. Il devient ainsi possible de dupliquer, sauvegarder ou même déplacer l'application pour la relancer sur un autre serveur. La suspension-sauvegarde-déplacement-redémarrage est un équivalent de la migration. Par abus de langage on ne parlera de la migration que pour les migrations transparentes précédemment décrites.

Les leviers internes ne fonctionnent que pour des sous-ensembles particuliers d'applications qui ont été spécifiées et dont le système de décision est au courant. Il n'existe pas de langage générique permettant d'exprimer les possibilités des applications ainsi que leur impact. Certaines modélisations, par exemple à base d'UML[7], permettent

d'exprimer la structure et certaines caractéristiques des applications. Ces représentations ne sont pas assez génériques pour modéliser l'ensemble des leviers ni leurs effets énergétiques.

Checkpointing Semblable à la sauvegarde, il s'agit de pouvoir sauvegarder l'état de l'application. Cette méthode permet de passer outre les limitations de la sauvegarde simple et de conserver un état global pour une application communiquante. La taille est aussi souvent plus petite car il s'agit ici de ne sauvegarder que ce qui est nécessaire.

Reconfiguration Certaines applications peuvent changer de comportement. Ainsi il est possible de changer le nombre d'instances d'un même serveur web pour être capable de répondre à une charge évoluant dans le temps par exemple. De manière plus générale, les reconfigurations mènent souvent à des changements de nombre de composants constituant une application. Un des modèles de telles applications est le modèle des tâches malléables[13]. Un modèle plus général est le modèle à composant[17] qui permet d'exprimer un plus grand nombre de propriétés et de reconfigurations.

3.3 Utilisation des leviers

Il existe un grand nombre de leviers autant du point de vue logiciel que matériel. Plusieurs questions se posent quand à l'utilisation de ces leviers au delà de leur but : Optimiser l'énergie et la performance par exemple.

Pour chaque levier il est nécessaire d'évaluer l'impact sur le système en utilisant les modèles du chapitre 2. Dans certain cas il est aussi nécessaire de pouvoir prédire le futur afin de pouvoir prévoir l'impact de l'utilisation d'un levier. La prédiction est nécessaire car beaucoup de leviers (démarrage, extinction par exemple) ont une latence assez grande. De plus certains effets sont à long terme, tels que les effets thermiques. Les méthodes de prédiction ont des problèmes de fiabilité qu'il est nécessaire d'intégrer.

Au delà de la problématique de la décision et des éléments sur lesquels elle se base, la stratégie de décision est elle aussi importante.

3.3.1 Temps entre les décisions

Dans beaucoup d'études, on considère l'existence d'un pas de temps qui correspond au temps entre deux décisions.

Dans [C2] nous avons évalué le comportement d'un système théorique optimal où la décision est remise en cause dès que les leviers ont agit. La granularité de décision est donc la même que les latences des leviers. Cette supposition est peu réaliste dans un système à grande échelle car elle consiste à reprendre des décisions globale de manière très rapprochée, ne laissant pas assez de temps au système de finir de prendre la décision précédente. Elle a l'avantage d'évaluer une sorte de comportement idéal où les décisions se font en flux continu.

Une méthodes plus classiquement utilisée consiste à avoir un pas de temps largement plus grand que celui lié aux reconfigurations et donc de reprendre régulièrement une décision. Ce pas de temps doit être déterminé en fonction des latences d'action du système mais aussi en fonction du dynamisme des applications.

L'autre grand modèle utilisé est le modèle événementiel. Ce modèle est hérité de l'âge des Grilles de calcul où les seuls moments où une décision était à prendre était lors du lancement ou de l'arrêt d'une tâche. Dans [J7] nous utilisons les informations sur le diagramme de tâche pour prendre à chaque début ou fin de tâche des décisions sur le fréquence des serveurs, le placement des tâches. Cette méthode fonctionne bien dans le cadre de l'étude citée car les profils des tâches de calcul haute performance évaluées sont relativement constants.

Avec les système actuels, où il est possible de faire des actions pendant la vie des tâches (changement de fréquence, migration) ou du fait de certains effets temporels (température), les évènements de début et fin de tâches ne sont plus suffisants. Dans un monde de services, où les tâches n'ont virtuellement plus de début ni fin mais seulement un profil de charge, ces événements font encore moins sens. Ainsi dans [C21] nous ajoutons aux événements classiques des détections de phases d'applications qui servent à détecter des changements de condition nécessitant des prises de décisions. Cette détection événementielle nous permet de gérer des applications qui sont limitées par le processeur pendant un temps, puis limitées par le réseau et de ne mettre la fréquence du processeur au plus bas que pendant la seconde phase.

3.3.2 Structure de la décision

Presque tous les systèmes de décision sont centralisés. Un serveur maintient à jour une vision globale du système, l'analyse, et prend une décision. Cette décision est alors exécutée.

Cette méthode est très efficace à petite échelle car les décisions ainsi prises le sont basées sur des informations fiables.

Dans un système à très grande échelle, plusieurs centaines de milliers de serveurs, le sur-coût de gestion devient rédhibitoire. Il devient impossible d'évaluer et de prédire l'effet des leviers et de concevoir un plan d'action global.

On se retrouve à avoir un modèle où on ne peut pas gérer tous les leviers à la granularité la plus fine. Il est alors nécessaire d'aller un pas plus loin dans la décentralisation.

Il existe deux pistes relativement compatibles :

— Systèmes hiérarchiques

— Système d'intention

Les systèmes hiérarchiques permettent de prendre la décision à une échelle raisonnable. On ne prendra la décision que par *blocs* de serveurs. Cela empêche certaines optimisations tels que la consolidation de plusieurs machines virtuelles entre les blocs mais permet de réduire de façon drastique les sur-coûts. Le véritable problème vient de l'optimisation qui se fait alors de façon non-coordonnée. Par exemple dans le cas où l'optimisation se fait à budget énergétique limité, il devient nécessaire d'allouer de

manière arbitraire des quotas d'énergie à chaque sous-partie qui les géreront de manière semi-indépendante.

Le système d'intention est quand à lui orienté du haut vers le bas. Il s'agit de considérer que les ordres venant du système de décision ne sont pas de la micro-gestion mais expriment des directions. Ainsi un ordre pourra être *réduire la consommation* auquel certains serveurs répondront en changeant de fréquence et d'autres en tentant de faire de la consolidation avec leur voisins. Un autre ordre pourrait être de réduire la température dans une zone et de la laisser monter dans une autre.

L'avenir de ces systèmes de décision sera sûrement un mélange entre des décisions locales (DVFS ou surchauffe par exemple) avec des coopérations hiérarchiques (pour la consolidation par exemple) couplés avec des systèmes d'intentions (gestion globale de la température).

3.4 Conclusion

Dans un monde idéal, l'utilisation de capteurs et de leviers ne serait pas utile. Heureusement le monde réel est plus complexe et ces deux éléments sont à la base de la gestion et de l'amélioration des centres de calcul.

Il existe plusieurs échelles pour ces deux éléments, de locale à globale. Bien souvent, l'échelon local est juste une passerelle pour un échelon global. Toute récupération d'information finit sur un serveur central, et toute utilisation de levier vient souvent du même serveur. Ce serveur a alors une vision précise de l'état du système. Pourtant, avec l'augmentation en taille des centres de calcul, cette méthodologie semble de moins en moins raisonnable. Une ébauche de gestion hiérarchique commence à apparaître. Les fédérations de centres de calcul distribuent la supervision et l'action, et certaines actions se font uniquement au niveau local (DVFS par exemple).

Pour passer à l'échelle des futurs centres de calcul, une méthode sera sûrement d'aller encore plus loin, potentiellement de proposer différents niveaux où des entités coopérantes travailleront de concert pour récupérer et traiter les informations pertinentes, mais aussi pour appliquer des intentions peu précises mais avec un impact local spécifique.

Au niveau plus local, des capteurs plus pertinents sont à concevoir. Des capteurs permettant d'évaluer plus finement les goulets d'étranglement par exemple permettraient de choisir de manière plus fine les compatibilités entre applications. Dans le paradigme de virtualisation, on considère que les applications n'interagissent pas. Pourtant du fait des goulets, la performance d'une machine virtuelle peut être impactée par l'exécution d'une autre VM. Cette supposition est approximative du fait de l'existence de ces goulets. De même des mécanismes permettant un apprentissage générique, sans dépendance à priori de l'infrastructure sont à préciser.

Enfin, une partie importante sur la temporalité est souvent peu prise en compte. Quelle granularité temporelle de mesure est pertinente, mais surtout est-il nécessaire que tous les capteurs partagent la même ? Cette problématique est un peu abordée au niveau des capteurs mais elle l'est très rarement au niveau de la décision où la question se pose dès que l'on considère une décision distribuée. Mais même dans le cas centralisé

actuel, des études sur l'impact du temps entre les décisions ou du choix des événements menant à une décision sont encore manquantes.

Chapitre 4

Les outils de l'évaluation de performance

> Donne-moi où je puisse me tenir ferme, et j'ébranlerai la Terre
>
> Archimède

La modélisation fine et complète d'un centre de calcul est difficile comme nous l'avons vu dans le chapitre précédent. En plus de la complexité intrinsèque il est nécessaire de gérer la problématique de l'échelle et ce, sur plusieurs points de vue :

- Nombre de ressources : On est en présence de plusieurs milliers de serveurs ;
- Nombre de leviers : Comme vu précédemment il y a des leviers au niveau matériel, au niveau logiciel ;
- Point de vue temporel : L'optimisation se fait de manière globale car trouver l'optimal à chaque instant ne garantit pas de trouver l'optimal global.

On considérera dans la suite que l'on gère un cloud composé de serveurs, et que ces serveurs seront définis par leur capacité mémoire et processeur. Ce dernier sera capable de DVFS. Comme vu en [97], on peut ensuite facilement généraliser à n'importe quel nombre de ressources fluides et rigides. De même les applications, ici modélisées par des machines virtuelles auront des besoins dans ces deux catégories évoluant au cours du temps.

De manière classique il existe trois méthodes pour étudier la performance d'un système : la résolution exacte, la simulation ou l'expérimentation.

De manière générale, les problèmes de placement de tâches et de configuration de serveurs sont NP-Complets[97]. La recherche de solution optimale est donc exponentielle en temps en fonction de la complexité du problème. Cette méthode est donc limitée à de petites tailles.

Pourtant cette méthode permet dans certains cas atteignables d'avoir des informations sur le comportement du système, mais aussi sur la qualité des approximations possibles.

Vu que les modèles exacts (tels que les modèles linéaires) ne permettent pas de résoudre complètement ce problème, il est nécessaire de l'attaquer aussi sous les deux autres angles complémentaires : simulation et expérimentation. Le premier permet de faire des tests rapidement avec une bonne couverture au prix de la précision du résultat. Le second permet d'avoir la précision mais demande un temps conséquent.

4.1 Programmation linéaire

Même si il n'est à l'heure actuelle pas connu d'algorithme polynomiaux pour résoudre les problèmes NP-Complets[40], il existe plusieurs méthodes permettant d'accélérer la recherche exhaustive. En effet on a une modélisation complète du système ainsi que des fonctions objectif claires. Il est ainsi possible par exemple de faire des recherches exhaustives dans des arbres de possibilités en utilisant une technique de *Branch and Bound*[63, 107] ou de modéliser ce problème par une technique de programmation linéaire. Cette dernière méthode est plus limitée dans son expressivité car elle ne permet pas d'exprimer tous les problèmes, mais elle présente l'avantage d'être plus efficace. Cette famille d'algorithme a une complexité en pire cas exponentielle mais une complexité polynomiale en moyenne.

4.1.1 Modélisation directe en programmation linéaire

Modéliser un problème par des techniques de programmation linéaire consiste à produire un ensemble d'équations affines séparés en deux catégories :

— Des contraintes (sous la forme d'équations ou d'inéquations)

— Un objectif qui est une expression à maximiser ou à minimiser

La limite de cette méthode est de ne pouvoir utiliser que des formulations par équations linéaires pour la modélisation. Il existe des méthodes pour linéariser des expressions non-linéaires. Ainsi une méthode classique est de transformer une inéquation non-linéaire convexe en une série d'inéquations linéaires. Plus la précision recherchée sera grande, plus le nombre d'équations nécessaires à assurer la linéarisation sera grand lui aussi.

4.1.1.1 Formulation linéaire des contraintes

La formulation linéaire se compose de variables d'états qui décrivent le système et son état. Ainsi une variable discrète d'allocation des tâches sur les serveurs e_{jh} est à 1 si la tache j est allouée sur la machine h et à 0 sinon. Une variable rationnelle telles que α_{jh} représente entre 0 et 1 la part des ressources de la machine h allouée à la tâche j.

Le tableau 4.1 montre les notations utilisées dans un cadre de ressources homogènes (i.e les consommations électriques des serveurs sont individuelles, mais les quantités de mémoire et les puissances de calcul des processeurs sont identiques).

En utilisant ces variables, on peut alors exprimer les lois présentées en Chapitre 2.

La modélisation des lois de 2.1 à 2.7 explicitent les lois physiques régissant les centres de calcul.

Nom	**Paramètre d'entrée**	**Type**	**Description**
J	oui	Entier	Nombre de tâches
H	oui	Entier	Nombre d'hôtes total
C_h^{min}	oui	Rationnel	Consommation en watts de l'hôte h lorsqu'il est inactif mais allumé
C_h^{max}	oui	Rationnel	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)
v_j	non	Binaire	La tâche j s'exécute (=1) ou est suspendue (=0)
α_{jh}	non	Rationnel	Fraction de CPU allouée à j sur l'hôte h
r_j	oui	Rationnel	Fraction de CPU demandé par la tâche j
m_j	oui	Rationnel	Fraction de RAM requis par la tâche j

TABLE 4.1 – Résumé des notations utilisées pour décrire un centre de calcul en utilisant une modélisation par équations linéaires.

Le fait qu'une machine soit allumée ou éteinte (loi 2.1) est modélisé par la formule 4.1.

La loi 2.2 qui exprime qu'une VM est allouée sur exactement un et un seul serveur à un moment donné est modélisée par les formules 4.2. Dans ce cas comme chaque e_{jh} a pour valeur 0 ou 1 et que la somme pour une tâche de tous ses e vaut 1, alors exactement un de ces e vaut 1 et tous les autres valent 0. On y retrouve aussi la mise en cohérence entre α et m qui sont les ressources allouées sur une machine et e qui est la présence d'une tâche sur une machine. Si e pour une tâche est nul sur un serveur, alors aucune ressource de ce serveur n'est allouée à cette tâche.

La loi 2.3 qui déclare qu'une VM est soit en train de s'exécuter soit arrêtée est modélisée par les équations 4.3. Elles déclarent que si une tâche est stoppée (v nul) alors elle ne consomme aucune ressource.

Les lois complémentaires 2.4 et 2.5 qui lient le fait qu'une tâche ne peut s'exécuter que sur une machine allumée sont modélisées par les équations 4.4 et 4.5. On remarquera que ces équations sont simplifiables en équation toujours vraies pour toutes les valeurs de (j, h) où $e_{jh} = 0$. En effet elles ne représentent des contraintes pertinentes que lorsqu'on analyse une machine et une des tâches s'y exécutant.

Enfin, les équations 4.6 et 4.7 modélisent la loi 2.6 respectivement pour les ressources fluides et les ressources rigides.

La loi 2.7 n'est pas modélisée directement mais par le fait de ne pas avoir un équivalent à α pour les ressources rigides.

$$\forall h \quad p_h \in \{0,1\} \tag{4.1}$$

$$\forall j \quad \sum_h e_{jh} = 1, \quad \forall j,h \quad e_{jh} \in \{0,1\}, \alpha_{jh} \leq e_{jh}, m_{jh} \leq e_{jh} \tag{4.2}$$

$$\forall j \quad v_j \in \{0,1\}, \quad \forall h \quad \sum_j \alpha_{jh} \leq v_j, \sum_j m_{jh} \leq v_j \tag{4.3}$$

$$\forall j,h \quad p_h \leq (1 - e_{jh}) + v_j \tag{4.4}$$

$$\forall j,h \quad v_j \leq (1 - e_{jh}) + p_h \tag{4.5}$$

$$\forall h \quad \sum_j \alpha_{jh}/r_j \leq v_j \tag{4.6}$$

$$\forall j \quad \sum_h m_{jh} \leq v_j \tag{4.7}$$

Lors d'une collaboration avec Henri Casanova, nous avons étudié[J1] ce jeu d'équation qui nous a permis d'avoir une solution optimale au problème de l'allocation de VM dans un cloud, puis de comparer cette solution optimale avec des heuristiques. A partir de ce jeu de base, il est possible de préciser la modélisation du centre de calcul. Ainsi nous avons complété[C2] cette modélisation avec les notions temporelles d'allumage et d'extinction de serveurs ce qui a montré que pour faire une optimisation globale il ne suffisait pas de faire des optimisations locale puis de migrer entre deux placements optimaux. Nous avons aussi étudié[D1] la modélisation linéaire du système de refroidissement des armoires de serveurs et montré qu'en tenant compte de la hiérarchie du système de refroidissement il était possible de réduire sa consommation électrique de 11% sans impact sur la performance.

Il existe plusieurs méthodes pour prendre en compte la temporalité et donc pouvoir gérer les migrations, allumages et extinctions des noeuds.

Une première méthode de modélisation consiste à considérer qu'il existe un nombre borné d'évènements (allumage, extinction, migration, changement d'états) et qu'il existe ainsi un ensemble d'équations et de variable représentant ces changements. Cette méthode permet de gérer un temps continu, mais complexifie fortement l'écriture des équations linéaires.

Une seconde méthode consiste à rajouter trois éléments :

— Rendre le temps discret
— Indicer toutes les variables par le temps
— Rajouter des équations de cohérence temporelles.

Le problème de cette méthode est qu'elle force à discrétiser le temps et à exprimer toutes les opérations (exécution, migration, allumage,...) en nombre de pas de temps.

Le premier élément est relativement simple, il s'agit ici de faire en sorte que chaque variable soit aussi indicée par le temps. Par exemple, e_{jh} qui représente la présence de la tâche j sur l'hôte h deviendra e^t_{jh}. Les équations de placement présentées ci-dessus reste alors valables pour chaque t

Le second force à manipuler des variables avec des t différents. Par exemple un hôte h est allumé au temps t si il était allumé ou en train de s'allumer au temps $t-1$:

$$p_h^t \geq p_h^{t-1} + allumage_h^{t-1} \tag{4.8}$$

De même si on considère que l'allumage prend exactement un pas de temps :

$$allumage_h^t + allumage_h^{t+1} \leq 1 \tag{4.9}$$

4.1.1.2 Formulation linéaire des objectifs

Les trois cas les plus simples sont lorsqu'on considère uniquement la performance et l'énergie. On peut choisir entre les objectifs suivants :

1. Borne sur la puissance maximale et maximisation de la performance minimale
2. Borne sur la performance minimale et minimisation puissance
3. Compromis : minimiser pour un certain λ l'équation suivante : $\lambda Performance + (1-\lambda)(1-Puissance_{norm})$

On considère la performance comme étant le ratio entre les ressources demandées et les ressources allouées. Cette caractéristique est naturellement normalisée. Pour que le compromis soit équilibré, il faut aussi normaliser la puissance.

La difficulté du choix de la formulation des objectifs est générale et ne découle pas de la nature linéaire de la modélisation. Dans les trois cas, il est difficile de fixer les valeurs arbitraires des seuils ou du lambda. Dans tous les cas, il s'agit d'explorer une partie du front de Pareto comme montré sur la figure 4.1. On défini le front de Pareto[79] comme l'ensemble des solutions pour lesquelles il n'existe aucune autre solution possible qui soit meilleure pour toutes les métriques. Ces solutions sont non dominées.

Dans [J9] nous avons proposé une méthode *floue* pour gérer de manière non arbitraire les méthodes bornées (i.e. les cas *1.* et *2.* ci-dessus). Il s'agit de trouver l'optimal pour une métrique particulière puis de fixer comme seuil pour cette métrique $X\%$ et optimiser sur l'autre objectif. Dans l'exemple de la figure 4.1 la solution choisie est la verte car une fois la contrainte sur la valeur de l'objectif 1 imposé, c'est la meilleure valeur pour l'objectif 2. Il s'agit toujours de fixer de manière arbitraire le coefficient de dégradation, mais il ne s'agit plus de fixer une valeur dépendante du problème pour un administrateur. Une étude plus complète des métriques multi-objectifs sera abordée dans la section 5.3.2.

4.1.2 Complexité temporelle

Le principal problème des méthodes exactes est le temps nécessaire à trouver un état optimal. En effet, le problème est NP-difficile comme la majorité des problèmes d'allocations de ressources[97]. Il est *trivial* de le prouver grâce à une réduction au problème du *bin-packing*.

Un des problèmes vient du fait du très grand nombre de variables et d'équations. En effet, on a un nombre d'équations et de variables de l'ordre de grandeur de $J \times H$.

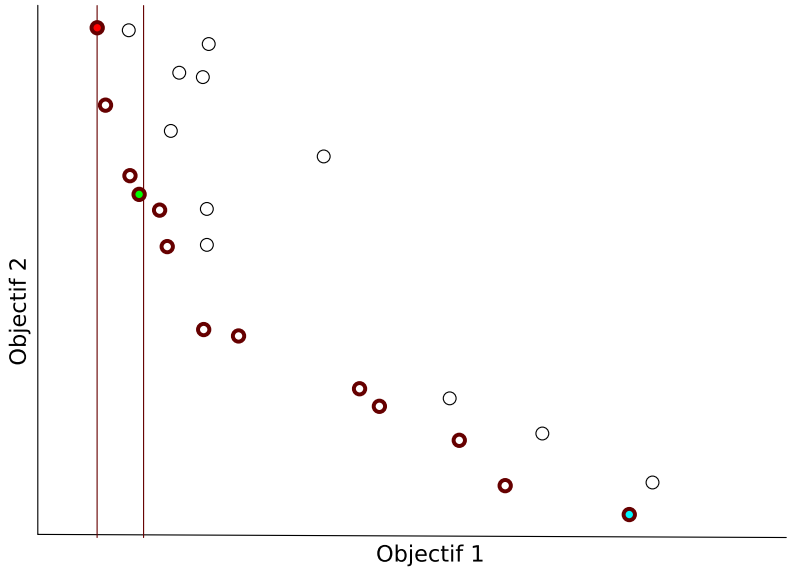


FIGURE 4.1 – Dans le cas multi-objectifs ils n'existe pas de solution optimale simple. Tous les cercles sont ici des solutions. Le front de Pareto représente toutes les solutions étant les *meilleures*, i.e. non dominées (en gras). La valeur rouge (resp. bleue) est la meilleure pour l'objectif 1 (resp. 2). En fonction du compromis entre les deux objectifs, toutes les valeurs du front de Pareto peuvent être choisies, par exemple la verte.

Dans le cas simple d'un placement statique (où il n'y a pas de considérations thermiques ni aucune évolution du système), nous arrivons à gérer un cas de 6 hôtes et 16 tâches avec un temps de calcul de l'ordre de 3 minutes[J1] en utilisant GLPK[67] sur un AMD Opteron 248 2.2GHz.

Lors du passage au temps, une méthode consiste à discrétiser le temps et à rajouter un indice pour chaque variable pour donner sa valeur en fonction du temps. Si on a v variables pour le problème statique et p pas de temps, on aura ainsi $v \times p$ variables résultantes. Il est aussi nécessaire de rajouter les équations de cohérence temporelle qui rajoutent elles aussi des variables supplémentaires.

Pour pouvoir étudier des cas plus réalistes, une des méthodes consiste à relâcher la contrainte entière. Cela permet d'utiliser des méthodes telles que l'algorithme du simplex à variables rationnelles qui est en pratique polynomial en fonction du nombre de variables et d'équations même si il est exponentiel dans le pire cas[94]. Le solveur utilise les mêmes équations, mais en autorisant les variables entières à avoir des valeurs rationnelles. On aboutit à une solution approchée qui est non réaliste. Dans ce genre de solutions, on a par exemple une tâche à moitié sur un serveur et à moitié sur un autre. Dans un système à très grande échelle (grand nombre de tâche par hôte) on se rapproche de l'optimal. Une machine avec une demi-tâche a plus d'importance relative qu'une machine avec dix tâches entières et une demi. Dans tous les cas, cette méthode est plus rapide et donne une borne inférieure à la solution optimale qui permet d'évaluer à gros grain la qualité

d'heuristiques. Toujours avec notre modélisation[J1] nous traitons le placement de 192 tâches sur 64 machines en une vingtaine de secondes. Pour comparaison, des heuristiques classiques de type best-fit prennent de l'ordre de la milliseconde dans ce cas.

En allant plus loin, en partant du modèle des tâches et machines et en les agrégeant il est même possible d'avoir des évaluations en temps constant, en groupant les tâches par catégories. Il s'agit de considérer les équations de tâches comme étant des équations de particules. Par un passage micro-meso il est alors possible d'avoir des équations globales indépendantes du nombre de tâches, comme pour un fluide dont on obtient l'équation d'écoulement sans connaître le comportement de chaque molécule. Dans [C7] nous avons montré qu'à grande échelle le comportement entre une simulation précise et une simulation fluide sont semblables malgré des temps de traitement diminués de plusieurs ordres de grandeurs pour la seconde méthode.

Une autre méthode de résolution approchée consiste à fixer certaines dimensions du problème (allumage des hôtes) pour faire une première optimisation, puis à fixer une autre dimension (fréquence des processeurs) et ainsi à itérer sur un sous-problème. On trouvera ainsi une solution approchée qui peut être un minimum local, mais avec un nombre de variables à optimiser à chaque instant bien plus faible.

4.1.3 Domaine d'application

D'un point de vue performance pure, il est impossible d'utiliser la programmation linéaire directement pour résoudre des problèmes réalistes de gestion de centre de calcul.

Le relâchement des solutions entières permet de donner une borne à la solution optimale. Plus le problème est grand, plus cette borne est proche de l'optimal. Cette méthode permet de tester des heuristiques et n'a pas vocation à être utilisée dans un cadre réel.

La méthode itérative de résolution par fixation de variable permet elle de trouver un minimum local. Cette solution sera donc moins bonne que l'optimale. Elle présente l'avantage de pouvoir être mise en place dans un centre de calcul réel et a un rôle plus classique d'heuristique.

Au delà de la performance pure, la formalisation linéaire du problème permet d'avoir un formalisme dans lequel on modélise le système dans son ensemble. On peut l'utiliser pour de la vérification d'autres mécanismes. Ainsi on peut l'utiliser comme formalisme commun pour tester les solutions de plusieurs heuristiques et les comparer.

4.2 Simulation

Il existe un grand nombre de simulateurs de centres de calcul. Les deux principaux sont SimGrid[22] et CloudSim[21]. Le premier a l'avantage de la précision de simulation et sa communauté très active, le second a l'avantage de la diffusion et de la facilité de rajouter des possibilités par des greffons. SimGrid vise expressément la simulation de centre de calcul orientés HPC. CloudSim est une nouvelle version de GridSim qui a la même optique de SimGrid. En plus de ces deux simulateurs il existe un grand nombre de

Simulateur	Avantage	Inconvénient
PeerSim [73]	Décentralisé	Pas de modèle de réseau, ni de virtualisation, documentation faible
GridSim [20]	Réseau, grille, tâches, ressources modélisées, bonne documentation	Pas de modèle de l'énergie, ni de la virtualisation
CloudSim [21]	Modèle de l'énergie, hérite de GridSim, gestion des VM	Gère seulement l'énergie pour l'implication monétaire
PlanetSim [42]	Décentralisé	Pas de métriques ni de modèle du réseau
SimGrid [22]	Passage à l'échelle, modèle réseau, métriques	Pas de modèle de l'énergie, du DVFS ou de la virtualisation
Dcworm [60]	Passage à l'échelle, modèle réseau, métriques, facile d'accès	Pas de modèle de l'énergie, documentation légère, ordonnanceur centralisé
MagateSimulator [54]	Passage à l'échelle, modèle réseau, métriques, ordonnanceur décentralisé	Pas de modèle de l'énergie, documentation légère,
Alea 3 [58]	Passage à l'échelle, modèle réseau, métriques	Pas de modèle de l'énergie, documentation légère, ordonnanceur centralisé

TABLE 4.2 – Comparatif des simulateurs de centre de calcul au début de nos travaux vers 2010.

simulateurs dédiés à certaines caractéristiques tels que NS/3 pour le réseau, HydraSim que nous avons développé pour les centres de calcul hybrides CPU/GPU ou RenewSim que nous développons à l'heure actuelle et qui s'intéresse plus particulièrement aux centres de calcul fonctionnant à l'énergie solaire et renouvelable en général.

4.2.1 État des lieux des simulateurs existants

Une grande difficulté pour l'étude tenant compte de l'énergie des centres de calcul est la nécessité d'intégrer les modèles décrits dans le chapitre 2.

Le tableau 4.2 montre les caractéristiques (positives et négatives) des simulateurs existants. Il illustre la complexité de la situation.

De manière générale il n'existe pas de simulateur largement utilisé permettant de simuler un centre de calcul en prenant en compte tous les leviers (migration, allumage, extinction, DVFS), en récupérant des informations énergétique précises (puissance et énergie consommées par tâche ou VM) et facile d'utilisation pour tester des configurations multi-sites ou enfin des effets complexes tels que les effets thermiques.

4.2.2 Amélioration des simulateurs

Dans les différents travaux que nous avons effectués, nous avons soit développé des simulateurs ad-hoc pour des cas particuliers (simulateur hybrid CPU-GPU *HydraSim*, simulateur de proportionalité de centre de calcul[C4, J11], simulateur de centre de calcul alimenté par panneaux solaires *RenewSim*) soit nous avons dans le cadre de collaborations participé à l'écriture de greffons pour combler les manques des simulateurs existants.

Nous avons principalement utilisé comme base CloudSim et DCWorms. Le second dérivant du premier dans ses versions antérieures a gardé une bonne extensibilité grâce au système de greffons.

Les différents greffons que nous avons développés sont les suivants :

Puissance Dans l'ensemble de nos travaux, afin de pouvoir évaluer les différents algorithmes, nous avons du en premier ajouter[D3, D4] les modèles de puissance ainsi que la métrique énergétique.

Machines Virtuelles Le second modèle nécessaire a été la gestion fine des machines virtuelles[D3, D4], en tenant compte de la sur-réservation de ressources et du ralentissement afférent mais aussi de l'hétérogénéité des architectures[J11] pour gérer aussi bien des architectures x86 que ARM en tenant compte des migrations entres les deux.

DVFS Le dernier ajout[J7] d'un point de vue de la puissance de calcul est le DVFS. En effet un des points différenciants entre Grilles et Cloud et la charge des machines qui ne sont pas soit saturées soit arrêtées pour ce second type de système. Pour avoir une évaluation réaliste et fine, nous avons implémenté dans CloudSim un plugin de gestion de gouverneur. Nous y avons implémenté les quatres gouverneurs classiques que l'on trouve dans le monde Linux : *Performance*, *PowerSave*, *OnDemand*, *Conservative*.

Thermique Dans le cadre du projet CoolEmAll, nous avons montré que la problématique thermique était importante[J9] car en tenant compte de la production de chaleur des serveurs dans la gestion des VMs il est possible d'en réduire la température maximale sans impact sur la performance des VMs ni la consommation électrique, ce qui permet de réduire la consommation énergétique du système de refroidissement. En fonction de la charge d'une machine, mais aussi de son mode et de sa position, nous avons développé[J12, J9] un greffon d'évaluation de la température des processeurs pour DCWorms. En effet, en fonction de la position d'une machine, la chaleur produite par celle ci va chauffer plus ou moins d'autres machines. Tenir compte de ce phénomène permet d'évaluer plus finement des algorithmes tentant de réduire la température maximum des machines, ce qui a un impact à la fois sur leurs courant de fuite (voir section 2.3.1) mais aussi sur l'infrastructure de refroidissement.

Multi-Cloud Plusieurs simulateurs basés sur GridSim sont capables de gérer des grilles multi-sites, et CloudSim basé lui aussi sur GridSim permet de gérer un site de Cloud. Nous avons donc fusionné[C19] la couche multi-site du simulateur

MagateSim[54] avec la gestion de machines virtuelles de CloudSim afin de pouvoir étudier les fédérations de centres de calcul (voir Section 6.1).

Métriques Enfin, la majorité des simulateurs venant du monde des grilles donnaient principalement comme métriques un temps total d'éxécution. Nous avons rajouté dans CloudSim d'autres métriques de QoS[J8] telles que la puissance, le dynamisme, la satisfaction des VMs ainsi que la robustesse. Dans le cadre du projet CoolEmAll, nous avons participé à l'élaboration et à la définition des métriques présentées en table 2.1 et à leur implémentation dans DCWorms[60].

4.2.2.1 Ajout du DVFS

La majorité des simulateurs existants sont basés sur des simulateurs de grilles de calcul comme SimGrid ou GridSim.

Dans une grille de calcul, les tâches sont le plus souvent seules sur leurs ressources (coeurs ou processeurs) et les utilisent à 100%. Ces simulateurs ont donc une vision simple de chaque ressource : *utilisée* ou *libre*.

Dans un centre de calcul la situation est plus compliquée. Il devient nécessaire de gérer plusieurs tâches sur chaque élément, chacune de ces tâches ne prenant qu'une fraction des ressources. De plus à un instant donné une ressource peut n'être utilisée qu'en partie.

C'est en utilisant cette caractéristique qu'il devient intéressant de faire de la consolidation, i.e. de rassembler plusieurs tâches sur une seule machine. Mais pour évaluer les véritables gains obtenus grâce à cette méthode il est nécessaire de savoir combien consommerait une machine peu chargée, et ce de manière réaliste. Comme nous l'avons vu en 2.2 le DVFS permet d'adapter la puissance consommée en fonction de la charge. Il est donc nécessaire que les simulateurs tiennent compte de ces changements de fréquence afin de comparer différentes méthodes.

Durant notre collaboration avec l'équipe de Rajkumar Buyya, professeur à l'Université de Melbourne en Australie, qui développe CloudSim nous y avons intégré et évalué[J7] le DVFS ainsi que les modèles énergétiques liés.

La figure 4.2 montre une simulation d'une charge simple de type montée en charge puis descente. Même avec un comportement de charge simple, un serveur moderne utilisant le DVFS classique *ondemand* a un comportement complexe.

La figure 4.3 montre une charge plus réaliste ainsi que la comparaison entre simulation et expérimentation (figure 4.4) pour le même gouverneur *ondemand*.

La difficulté d'implémentation du DVFS vient du changement à apporter au temps nécessaire à effectuer les tâches. Dans un simulateur classique, lorsqu'une tâche commence, en interne le simulateur prédit sa fin et ajoute un événement à cet instant. Dans le cas du DVFS, l'instant de cet événement de fin peut changer en fonction des changements de fréquence du processeur.

En plus de ce recalage temporel il est aussi nécessaire d'évaluer la consommation électrique en fonction du mode.

D'un point de vue utilisation, on obtient quelque chose de très précis mais au prix

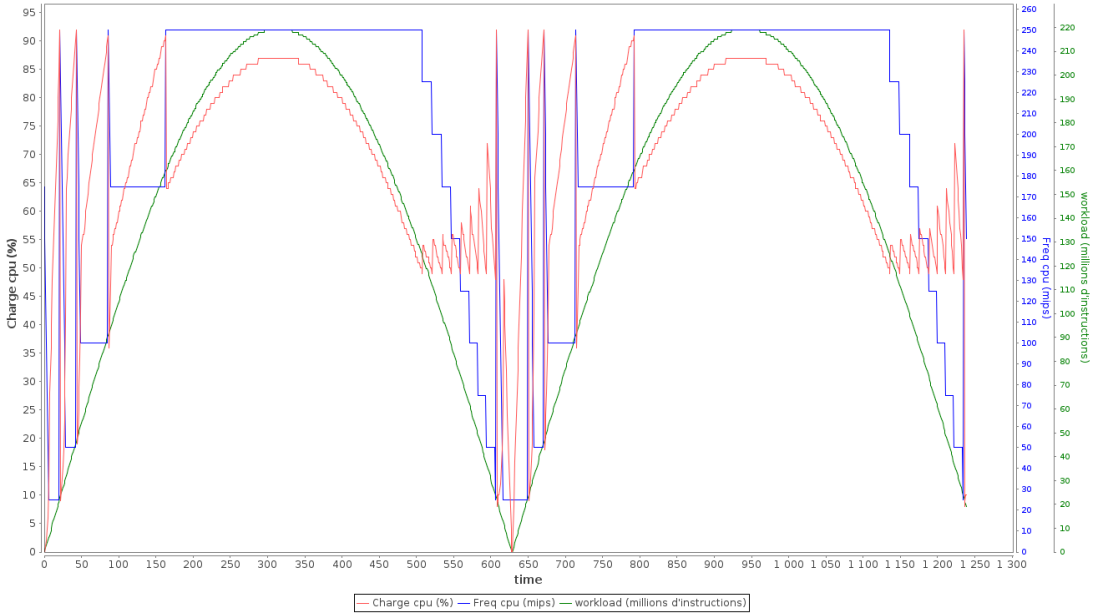


FIGURE 4.2 – Simulation d'une charge variable (en vert). Le simulateur utilise le gouverneur *ondemand* pour gérer la fréquence (en bleu). La simulation du pourcentage processeur utilisé (en rouge) dépend de la charge et de la fréquence.

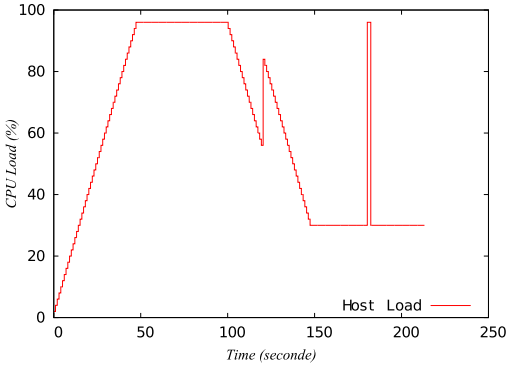


FIGURE 4.3 – Charge complexe pour illustrer le DVFS dans le cas où la fréquence est fixée au maximum.

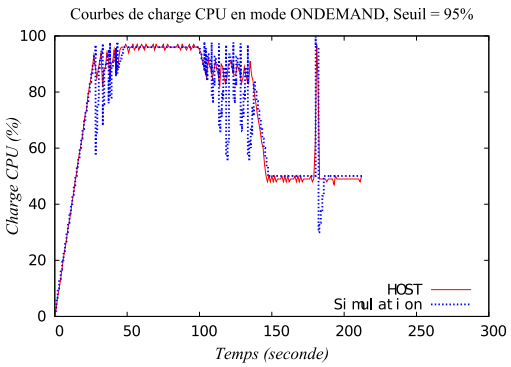


FIGURE 4.4 – Comparaison entre simulation et expérimentation du gouverneur ondemand avec la charge de la figure 4.3.

de la lenteur de la simulation. En effet il devient nécessaire de passer outre l'hypothèse simplificatrice des simulateurs classiques, qui est de se concentrer uniquement sur le début et la fin des tâches. Dans un cadre de cloud les tâches et leurs besoins évoluent au cours du temps, et il devient plus difficile de faire de grand sauts en avant lors de la simulation. On passe ainsi presque d'une simulation événementielle à une simulation temporelle.

Cette capacité des simulateurs nous a par exemple permis d'évaluer des algorithmes classiques de sélection de "fréquence optimale" à base de diagrammes temporels et de les comparer avec des gouverneurs. Nous verrons en Section 6.2.1 que ces ajouts nous ont permis de comparer la méthode optimale avec les gouverneurs classiques.

4.2.3 Les simulateurs ad-hoc

Nous venons de voir que la problématique des simulateurs génériques est vaste et complexe. Pour avoir un résultat fin il est nécessaire de tenir compte de beaucoup d'éléments qui ont un grand coût d'utilisation. Pour certaines études précises il peut être intéressant d'implémenter un simulateur ad-hoc qui aura un coût d'utilisation très faible au prix d'une expressivité elle aussi très faible.

L'objectif de ces simulateurs ad-hoc est alors différent. Il n'est plus d'avoir une vision la plus précise possible d'un phénomène, mais d'avoir une évaluation d'une borne sur l'optimal. Comme dans [J1] en relâchant la contrainte entière sur les méthodes de résolution d'équations linéaire il était possible d'obtenir une borne inatteignable mais permettant d'avoir un positionnement d'heuristique par rapport à l'optimal, en utilisant des simulateurs ad-hoc on peut rapidement évaluer un système idéalisé.

Le simulateur HydraSim [1] avait pour objectif d'évaluer des algorithmes de gestion de centres de calcul hybrides CPU/GPU.

Le simulateur[C4] d'architecture hétérogènes avait pour but d'évaluer l'intérêt d'utiliser des architectures très hétérogènes pour atteindre un centre de calcul proportionnel tel que décrit en début du chapitre 3.

Enfin, le simulateur *RenewSim*, en cours de développement dans la thèse d'Inès, a pour but de co-simuler des centres de calcul, des systèmes de stockage et de production électrique principalement renouvelable. Le but de ce simulateur étant d'avoir des modèles d'une complexité cohérente entre les différents systèmes, il nous a semblé préférable d'adopter une architecture modulaire, en prévoyant de pouvoir plus tard remplacer la partie simulation de centre de calcul par un simulateur "classique".

Dans la pratique ces simulateurs ont été très rapides à développer et ont permis de vérifier que les sujets étaient effectivement intéressants. La problématique la plus importante concerne alors l'évolution de ces simulateurs. Comme dans beaucoup de jeux de sociétés[24, 2, 45], l'important est de trouver le point de bascule.

Dans [C4], le simulateur est capable de prendre une charge web et d'en déduire la consommation électrique d'un centre de calcul fortement hétérogène (Raspberry Pi, Intel Atom, Intel I7). Les coûts d'allumage, extinction, migration sont ignorés. Seul le

1. `http://sourceforge.net/projects/hydrasim/`

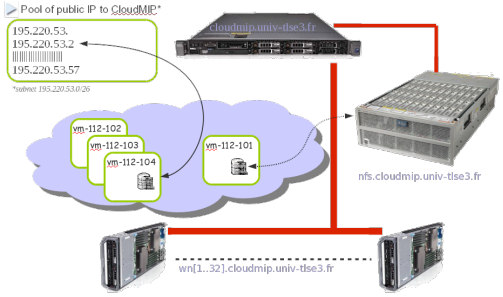


FIGURE 4.5 – Architecture de la plate-forme de cloud CloudMip

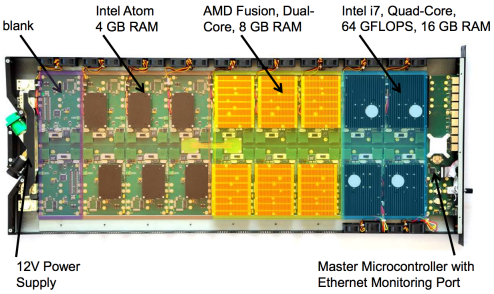


FIGURE 4.6 – Recs : Serveur haute densité contenant jusqu'à 18 modules

modèle de puissance en fonction de la charge est précis. Pourtant cette étude a permis de montrer qu'un centre de calcul fortement hétérogène permet d'atteindre une certaine proportionnalité.

Dans [J11], nous avons étendu ce simulateur pour prendre en compte l'énergie nécessaire à allumer et éteindre les serveurs. Grâce à cette précision accrue du simulateur il nous est possible de mettre en relation la dynamique de la charge (qui est forte dans l'exemple présenté dans l'article) avec les constantes de temps d'allumage et d'extinction des serveurs.

Pourtant, nous avons choisi de ne pas rajouter des éléments de contention lors des transferts réseau liés aux migration de VMs, ou de gérer précisément chaque événement (arrivée/traitement d'une requête). En effet, rajouter trop d'éléments nous forcerait à transformer ce simulateur ad-hoc en simulateur générique et donc à ré-implémenter ce qui existe déjà et qui a été présenté en section précédente.

La problématique devient d'être capable d'évaluer la perte de précision, le gain d'expérimentation et de développement. La situation est semblable lors du choix entre plate-forme d'expérimentation réelle et simulation. La méthode que nous suivons habituellement dans l'utilisation des simulateurs ad-hocs est de passer à la validation sur plate-forme réelle à des fins de validation.

4.3 Les plate-formes

De manière générale, au delà de la simulation il est nécessaire de valider grâce à des expérimentations réelles. Dans le cadre de ces recherches, j'ai pu avoir accès à plusieurs infrastructures matérielles, chacune avec sa personnalité propre.

Grid'5000 [2]. L'infrastructure la plus connue (voir Section 1.3.2). Grid'5000[14] a l'avantage de l'échelle mais aussi de pouvoir être reconfigurée à tous les niveaux pour s'adapter à l'expérience.

2. `https://www.grid5000.fr`

CloudMip [3]. Il s'agit d'un ensemble de 32 lames capables d'accueillir 256 machines virtuelles et géré par l'IRIT (Figure 4.5). Il s'agit d'un environnement proche d'une plate-forme de production. Même si les machines virtuelles contiennent du code académique, CloudMip permet de faire des tests in-situ.

RECS . La plateforme la plus expérimentale. Il s'agit d'un serveur haute densité (1U, moins de 5cm de haut). Un RECS[9] peut recevoir jusqu'à 18 modules indépendants. Dans notre cas le panachage de modules est montré en Figure 4.6. Il permet très facilement de faire des tests de systèmes très hétérogènes et dont les éléments ont des impacts les uns sur les autres.

4.3.1 Intergiciel

Chaque infrastructure physique est gérée par un intergiciel. Ce dernier gère les tâches (démarrage, migration,...) ainsi que l'état de chaque élément de l'infrastructure physique (machines allumées, éteintes) et gère aussi la supervision.

Grid'5000 utilise OAR[C3] avec un grand nombre d'outils supplémentaires tels que *kwapi*[88] qui permettent d'obtenir en temps réel la consommation électrique des machines sur les sites qui en sont capables. CloudMip utilise comme intergiciel de gestion de cloud OpenNebula[39] et depuis peu est passé à OpenStack[78].

Mais ces intergiciels ne sont pas doués d'intelligence. Les algorithmes utilisés pour décider où lancer les tâches ou machines virtuelles sont souvent de simples First-Fit. Une tâche sera lancée sur le premier serveur libre. Une des rares possibilités est celle de OAR qui est capable, lorsque des serveurs ne sont pas utilisés pendant longtemps, de les éteindre.

Dans le projet GreenNet[C5] puis dans le projet CoolEmAll[J2] nous avons décrit comment dans un système réel l'intergiciel devrait adapter le système en fonction de la surveillance de ce dernier. Le principe de base est celui de la boucle de rétroaction décrite en Section 1.2.3 et montrée en figure 4.7 plus précisément dans le cas de Green-Net.

Enfin, lors de la thèse de Landry[26], nous avons développé MREEF (Multi-Resource Energy Efficient Framework[J10]) qui est un intergiciel d'optimisation énérgétique. Il fonctionne sur le principe d'une boucle autonome (voir Section 1.2.3). Il gère uniquement les tâches une fois qu'elles sont en train de s'exécuter localement sur les serveurs. Il est donc complémentaire des intergiciels plus classiques que sont OAR, OpenStack, et OpenNebula.

4.3.2 Limites des expérimentations

Plusieurs difficultés limitent l'utilisation d'expériences réelles.

La première est le temps nécessaire à faire une expérimentation. Au delà de la difficulté de déployer cette expérimentation, le temps passe en 1 pour 1. En simulation, il est possible d'aller plusieurs ordres de grandeur plus vite.

3. `http://cloudmip.univ-tlse3.fr`

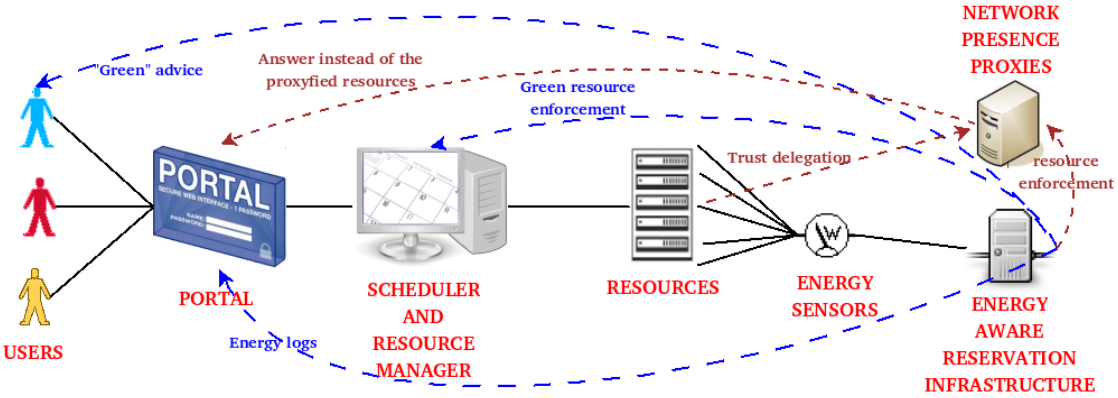


FIGURE 4.7 – Dans le projet Green-Net le système de décision prends des ordres informés des utilisateurs, récupère des informations de la plate-forme et utilise ces données pour atteindre les objectifs des utilisateurs

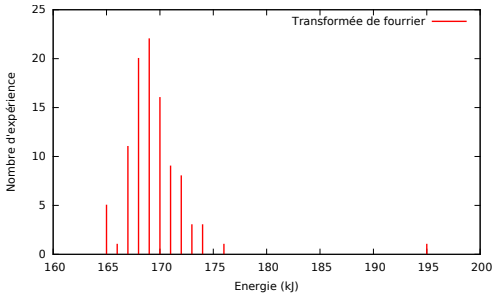


FIGURE 4.8 – Répartition de l'énergie de 100 exécutions identiques de FT

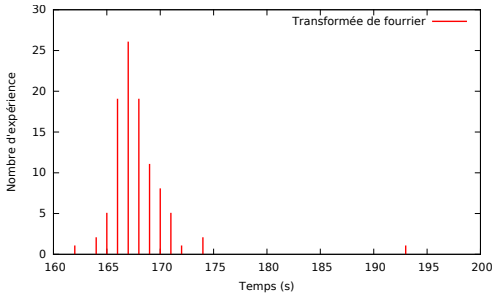


FIGURE 4.9 – Répartition du makespan de 100 exécutions identiques de FT

Mais cet écueil est renforcé par les problèmes de reproductibilité. Par exemple, lors d'une expérience (qui devrait être) simple de l'exécution d'une transformée de Fourrier sur 4 serveurs les mesures de temps et d'énergie consommées ont une forte variabilité alors que l'ensemble des 100 expériences ont été réalisées sur les mêmes serveurs.

Les Figures 4.8 et 4.9 montrent la répartition respectivement de l'énergie et du temps d'exécution. Pour l'énergie on a une moyenne de $170kJ$ avec un écart type de $3.3kJ$. Pour le temps on a respectivement $168s$ et $3.2s$. Une analyse rapide des données donnerait l'impression qu'il y a un corrélation forte entre l'énergie et le temps.

Pourtant l'étude de la corrélation entre ces variables est complexe. Comme le montre la figure 4.10, pour un même temps (par exemple 167s) on a des énergies assez différentes (4kJ, de 167kJ à 171kJ) et ce, sans même tenir compte des évènements anormaux (une expérience a mis $193s$ et consommé $196kJ$).

Pour obtenir une validité statistique suffisante, il a été nécessaire d'exécuter cette

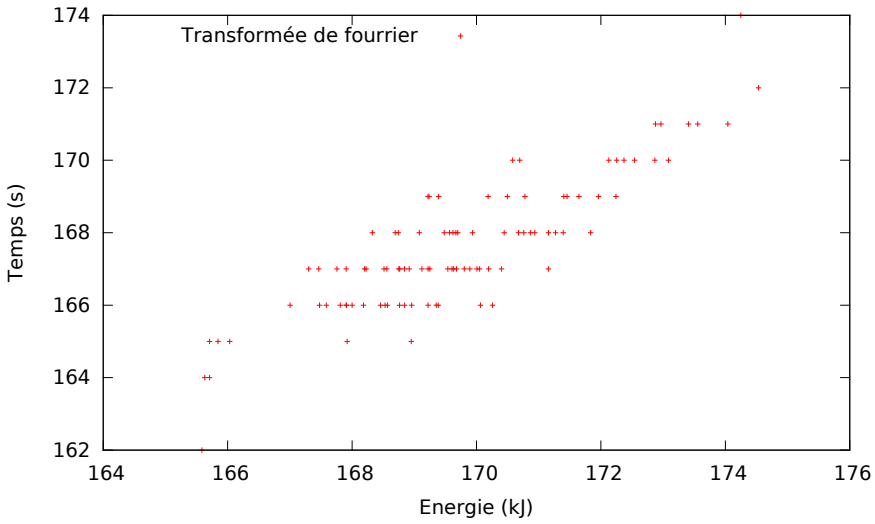


FIGURE 4.10 – Relation entre temps et énergie totale. La donnée aberrante visible sur les figures 4.9 et 4.8 a été enlevé pour améliorer la visibilité

expérience 100 fois ce qui rallonge d'autant le temps nécessaire à faire aboutir les expérimentations et les rends plus difficile à reproduire.

De manière plus générale, mesurer finement est problématique. Lors de la thèse de Léandro[D2] nous avons par exemple dû modéliser le comportement des wattmètres de façon à obtenir des séries temporelles de mesures de puissances électrique qui puissent être corrélées avec des séries de mesures de charges sur les serveurs par exemple. Plusieurs phénomènes existent : latence variable de la suite de mesure, latence variable entre les mesures, valeurs renvoyées en fait non mesurées,...

4.4 Conclusion

De manière assez classique, pour étudier de manière complète un problème il est nécessaire d'utiliser des outils de trois types : Modélisation mathématique, simulation et expérimentation.

Dans le cadre grande échelle dans laquelle nous nous plaçons, il est nécessaire de faire des relaxations pour la modélisation mathématique, le coût de résolution exacte étant trop grand.

Pour la simulation, au début de ces recherches, les simulateurs de centre de calcul avaient des capacités très disparates. Toute une série de travaux, dont les notres, ont permis de rendre ces outils plus adaptés au contraintes de la virtualisation, mais aussi de l'énergie et des leviers matériels.

Enfin, la partie expérimentale souffre de l'utilisation de matériel dédiés à d'autres usages. Dans les centres de calcul, les wattmètres servaient historiquement à détecter

des défaillances, et donc avaient une granularité temporelle de l'ordre de la minute, et une précision de quelques watts. Pour comprendre l'impact de décisions, il est nécessaire d'avoir une meilleure précision de plusieurs ordres de grandeurs pour ces deux granularités. Enfin, il faut remarquer qu'il ne nous a pas été possible de mener des expérimentation en sites de production, ni même d'avoir des données provenant de ce type de sites. L'augmentation de l'intérêt pour la recherche en économie d'énergie pour les centres de calcul devrait à terme permettre plus facilement de faire des expérimentations sur sites réels.

Une erreur communément commise consiste à lier directement économie d'énergie et performance. En effet, augmenter la performance d'une application permet souvent de faire des gains énergétiques. Pourtant, comme l'a montré la variabilité de la consommation énergétique d'applications ayant le même temps, à petite échelle ces deux métriques ne sont pas aussi simplement liées.

Chapitre 5

Résolution approchée par heuristiques

> La vie est vraiment simple, mais nous insistons à la rendre compliquée
>
> Confucius

Le problème principal que l'on étudie est le problème du positionnement au cours du temps des machines virtuelles. En effet, les autres leviers en découlent. Si on ne place aucune machine virtuelle sur un serveur, il est possible de l'éteindre. Si les VMs placées sur un serveur ne consomment pas toutes ses ressources de calcul ou de réseau, le DVFS ou l'ALR sont utilisables.

Comme décrit précédemment, on va séparer en deux étapes ce verrou scientifique complexe :

— Le placement initial des VMs
— La temporalité

5.1 Les heuristiques classiques de placement

Le point clé des différents algorithmes est de pré-trier les éléments à considérer pour réduire le coût de recherche de solutions et ne pas être exponentiel comme dans le cas de la recherche exhaustive.

Les grandes catégories sont les algorithmes :

— **Gloutons** First Fit, Best Fit, Round Robin, Aléatoire
— **VectorPacking** Ce bin-packing multi-dimension permet de gérer les différentes ressources de manière plus efficace
— **Algorithme génétique**

Dans tous les cas, on considère avoir deux listes : les tâches et les serveurs.

5.1.1 Algorithmes Gloutons

Il s'agit simplement de parcourir la liste des tâches et de trouver pour chacune un serveur pour la recevoir. La caractéristique principale est de ne jamais remettre en question les allocations précédentes. Dans ces algorithmes, on va considérer avoir une liste triée de serveurs lorsque ces derniers sont hétérogènes. Il s'agira souvent d'un tri en fonction de leur efficacité énergétique dans notre contexte. Nous verrons dans la section 5.3.1 que le tri des tâches est important par rapport à la qualité des placements.

Aléatoire Pour chaque tâche, on tire aléatoirement un serveur et on y alloue la tâche si il y a assez de place. Cet algorithme sert surtout à avoir une référence de comparaison.

FirstFit Pour chaque tâche, on parcourt la liste des serveurs et on alloue au premier possédant assez de place. Cet algorithme a l'avantage d'être rapide et de laisser un peu de ressources libres sur les serveurs. En revanche il n'est pas très bon pour consolider les tâches sur un faible nombre de serveurs et donc consommer peu d'énergie.

RoundRobin Pour chaque tâche, on cherche un serveur où le nombre de tâches est minimal. Cet algorithme est assez bon pour résister à des fautes de serveurs car dans ce cas là, peu de tâches vont êtres impactées. En échange sa consommation électrique est la plus forte car il allume un nombre maximal de machines.

BestFit Pour chaque tâche, on cherche un serveur où les ressources restantes après allocation sera minimale. Cet algorithme donne une très bonne consolidation et sera donc très efficace d'un point de vue consommation électrique. Par contre, dans un cas où les tâches sont dynamiques, elles auront du mal à obtenir plus de ressources.

Ces algorithmes sont fortement étudiés dans la littérature et ont un comportement prévisible. Ainsi la complexité temporelle est de $\mathcal{O}(J \times H)$

On peut voir sur la figure 5.1 l'importance du placement. Cette figure montre les différents algorithmes en considérant que les deux ressources sont rigides pour simplifier. Les *best-fit* et *fist-fit* avec besoin processeurs fluides auraient placés les tâches 1, 3 et 4 sur la machine 1, et la tâche 2 seule sur la machine 2. La charge processeur demandée aurait alors été de 110% sur la machine 1 et ses tâches auraient alors été ralenties.

Les deux tris que nous allons utiliser sont par besoins en processeur décroissant et par besoin en mémoire décroissant. Une explication de l'importance et de l'impact des tris se trouve en section 5.3.1.

5.1.2 Algorithmes de VectorPacking

Le vector packing est un cas particulier du problème du bin-packing qui consiste à trouver un arrangement d'articles pour occuper un minimum de boites. Le vector packing est la généralisation à plusieurs dimensions de ce problème. Ici les boîtes sont les serveurs et les articles sont les tâches.

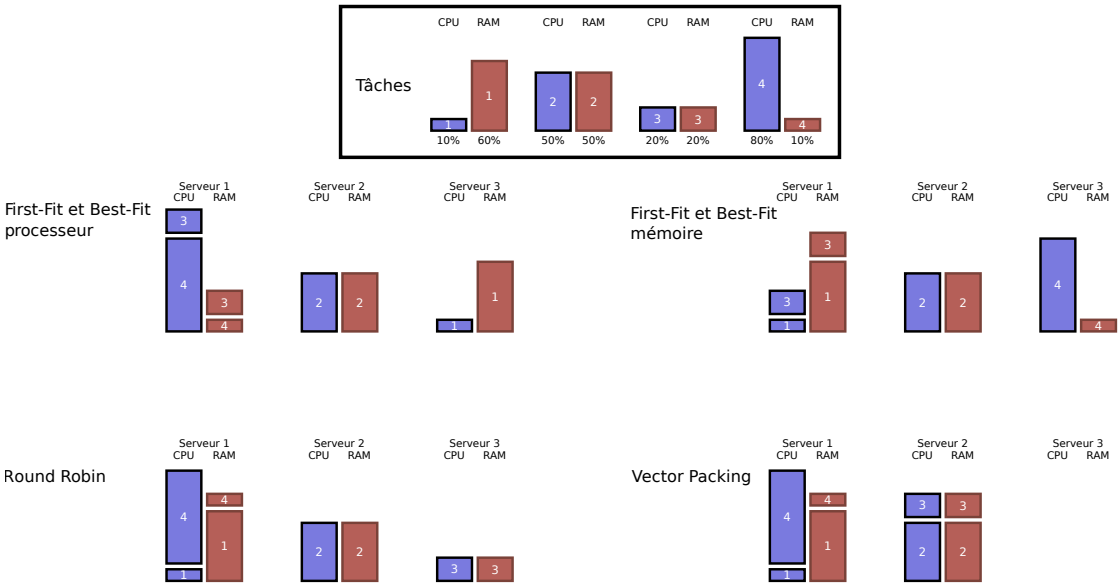


FIGURE 5.1 – Exemple de placement sur des machines identiques pour les algorithmes gloutons en considérant que les besoins mémoire et processeurs sont rigides.

Nous utilisons (par abus de langage) le terme de vector packing pour l'algorithme décrit ci dessous de par la multi-dimentionalité nécessaire : ressources rigides (mémoire) et fluides (processeur). Le but de réduire au maximum le nombre de boites est lié au fait que l'on va essayer d'éteindre un maximum de serveurs pour économiser de l'énergie.

Notre vector packing[J1] consiste en un algorithme glouton qui choisit pour chaque tâche un noeud ayant les caractéristiques suivantes :

— Le serveur est attractif, ie efficace d'un point de vue énergétique ;
— Rajouter la tâche ne mènerait pas à une surcharge ;
— Le serveur est déjà allumé et exécute déjà au moins une tâche ;
— La tâche rétablit l'équilibre dans les ressources.

Dans ces travaux, l'idée sera de généraliser les algorithmes gloutons classiques, en triant les tâches en fonction de chacune de leurs ressources. Dans notre cas, une liste triera les tâches en fonction de leur besoin mémoire, une autre en fonction de leur besoin processeur. Pour chaque machine, on va alors tenter de l'équilibrer, prenant dans la liste *mémoire* si les tâches qui lui sont allouées ont plus de besoins en calcul qu'en mémoire, ou dans l'autre liste dans l'autre cas. La figure 5.1 montre un exemple de ce placement qui est beaucoup plus efficace[J1] d'un point de vue énergétique que les autres algorithmes gloutons.

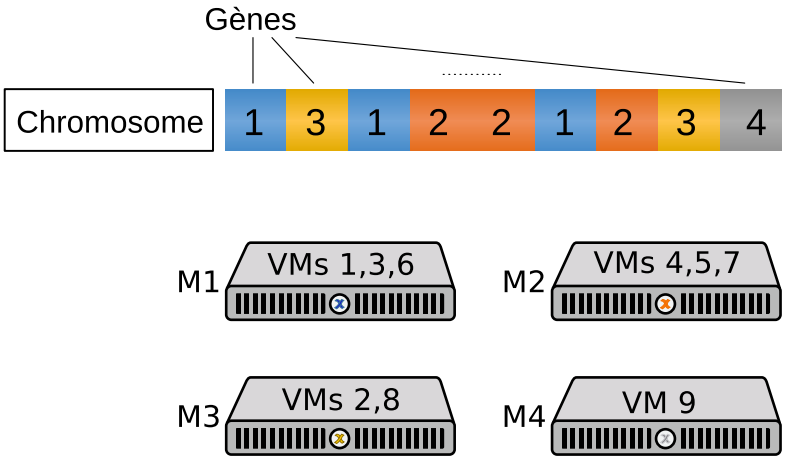


FIGURE 5.2 – Un chromosome est composé de gènes. Chaque gène représente le numéro du serveur sur lequel est alloué une machine virtuelle. Dans cet exemple, la VM n°2 est allouée sur le serveur $M3$

5.1.3 Algorithme génétique

Un algorithme génétique[72] est une meta-heuristique d'optimisation se calquant sur le principe de l'évolution des êtres vivants. Les générations d'êtres vivants vont croiser leurs patrimoines génétiques afin, par sélection naturelle, de produire des individus les plus efficaces possibles. De même les algorithmes génétiques vont manipuler un grand nombre d'individus (des placements dans notre cas) appelés *chromosomes*, les croiser et les faire muter en ne gardant que les meilleurs sur plusieurs générations.

Un algorithme génétique (Algorithme 1) va avoir besoin des éléments suivants :

— Une population initiale de chromosomes
— Des opérateurs de génération de chromosomes (croisement, mutation)
— Une fonction d'évaluation (*fitness*)

Algorithm 1 Algorithme génétique

1: Pop=Initial Population(N)
2: **while** NbIteration $\leq$ MaxIteration **do**
3: $Pop = Pop \cup \{mut(x) \mid x \in Pop\} \cup \{Croisement(x, y) \mid x, y \in Pop^2\}$
4: $Pop = Pop.best_{fitness}(N)$
5: **end while**

Les algorithmes génétiques permettent d'explorer des espaces de solution, d'une manière un peu différente des algorithmes de recuits simulés. La différence est que les algorithmes génétiques permettent de chercher en parallèle autour de plusieurs solutions pour les améliorer, et potentiellement lorsque les solutions comportent des parties indépendantes de les croiser pour obtenir des solutions plus efficaces.

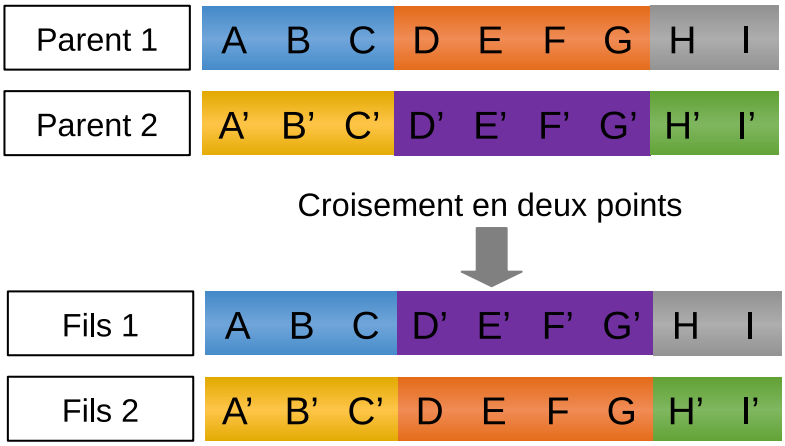


FIGURE 5.3 – Opération de croisement entre deux allocations de machines virtuelles

D'un point de vue modélisation, on va représenter un chromosome comme une suite de gènes, chacun étant la position d'une machine virtuelle sur une machine physique (Figure 5.2).

L'opération de mutation consiste à changer pour une des machines virtuelle sa destination.

L'opération de croisement (Figure 5.3) consiste à prendre une sous-partie de l'allocation d'un des deux chromosomes et à l'utiliser pour remplacer l'allocation des mêmes machines virtuelles dans l'autre chromosome.

La fonction de *fitness* est composée de deux parties. Une première partie consiste à vérifier que l'allocation est réaliste. En effet le coté aléatoire de cet algorithme peut produire des individus non viables. Par exemple le total de la mémoire allouée par les VMs sur un serveur peut dépasser sa capacité. La seconde partie est liée à une évaluation des métriques discutées dans la section 2.3.1.

5.2 Du placement au re-placement

Comme vu en section 1.2.3, certains événements temporels ou structurels (début ou fin de jobs, changement des besoins en ressources des tâches) peuvent provoquer une réallocation des tâches. Cette réallocation, grâce à la nouvelle situation a pour but d'améliorer les métriques.

5.2.1 First-Fit

La ré-allocation se fera en deux étapes. La première sera de trouver l'hôte le plus chargé pour distribuer ses tâches sur les autres serveurs en utilisant l'algorithme First-Fit. La seconde étape sera de prendre le serveur le moins chargé et n'ayant pas reçu de migration pour essayer de le vider lui aussi.

L'idée est de garder la notion de non-remise en question tout en essayant de gérer les deux problèmes principaux pouvant mener à un re-placement : une surcharge d'un serveur suite à une augmentation des ressources demandées par une tâche ; une sous-charge suite à l'arrêt d'une tâche.

L'ajout de tâche sera lui géré comme dans le cas statique.

5.2.2 MonteCarlo

La méthode de Monte Carlo est une technique probabiliste utilisée pour calculer des valeurs numériques. Dans notre cas, il va s'agir d'un abus de langage pour désigner une méthode proche de la création de la population initiale de l'algorithme génétique précédemment expliqué.

Ici, un grand nombre d'utilisation des leviers possibles vont être tirés au sort puis leur effet va être évalué en utilisant les métriques vues en 2.3.1. On gardera la meilleure solution.

5.2.3 VectorPacking

Il effectue un ensemble de migrations en partant des tâches les plus volumineuses, séparées en deux listes de manière à contrebalancer le déséquilibre sur les hôtes destinations. Ensuite, il fait 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 VMs dans le plus petit sous ensemble de machines physiques, pour ensuite équilibrer la charge parmi les machines choisies.

5.2.4 Algorithme Génétique

On utilise le même algorithme que précédemment mais en tenant compte du surcoût de migration dans la fonction de fitness.

5.3 Validation

Le but de cette section est de montrer les différents éléments ayant une importance lors de la construction d'une heuristique. En effet dans les algorithmes gloutons, la pertinence des tris est souvent prépondérante dans la qualité du résultat par exemple.

5.3.1 De l'importance des tris

Les algorithmes gloutons tels que First-Fit, Best-Fit et Vector-Packing utilisent les tris des serveurs et des tâches pour faire leur placement.

Nous allons étudier plus particulièrement le tri des tâches. On considère ici que les machines sont homogènes, la seule différence étant leur profil de consommation énergétique. De manière naturelle, les deux tris possibles pour les tâches sont le tri en fonction

de leurs besoins mémoire et de leurs besoins en calcul. Le cas du vector packing est particulier car il gère deux listes triées en fonction de ces deux besoins et alloue en fonction des déséquilibres.

Greedy-1 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 mémoire descendante). L'algorithme *Greedy-2* est aussi un algorithme First Fit, mais celui-ci triera les tâches par besoins CPU descendant. Les algorithmes que nous appellerons, *Greedy-3* et *Greedy-4* sont respectivement les versions best fit des algorithmes *Greedy-1* et *Greedy-2*. Enfin, *EARESALLOC* est l'algorithme de vector packing.

5.3.1.1 Cadre expérimental

Comme nous avons vu dans la section 2.3.1, il existe plusieurs métriques. Dans cette section nous allons nous concentrer sur la puissance instantanée ainsi que la performance.

La puissance instantanée (Watts) représente la qualité énergétique du placement car il s'agit d'une situation stationnaire.

La performance sera évaluée par la satisfaction (Yield) qui est le ratio entre les ressources processeur demandées par les tâches et les ressources qu'elles ont obtenues. Une tâche demandant 50% d'un processeur et n'obtenant que 25% aura donc un Yield de $\frac{1}{2}$. Cette valeur normalise la satisfaction des tâches et est sans dimension.

Les heuristiques décrites dans la partie précédente seront comparées avec la solution optimale obtenue par résolution du problème linéaire (*MILP*) pour la plus petite instance. Dans le cas des instances plus grandes, on se limitera à la version où la contrainte entière est relachée (*LPBOUND*) et où toutes les variables seront considérées comme rationnelles. Ce dernier algorithme donnera une borne hors d'atteinte mais qui permet de relativiser les différences entres les différentes heuristiques.

Dans la suite, les trois processus d'optimisation décrits en [J1] vont être étudiés :

— Optimiser la consommation sans dégradation de performance ;
— Optimiser la performance avec une borne sur la consommation ;
— Optimiser conjointement performance et consommation.

Les tâches utilisées pour la validation sont des tâches synthétiques générées suivant les profils proposés dans [97]. Il s'agit d'une charge simple dont les besoins en ressources mémoire et processeur suivent chacun une loi normale indépendante. Les Figures 5.4, 5.5, 5.6 et 5.7 montrent les agrégations de trois types d'expériences, à charge faible, moyenne et forte. Seul les résultats moyennés sont montrés car les comportements sont similaires dans les trois cas.

5.3.1.2 Expérimentation

Dans une première étape, la Figure 5.4 montre la puissance consommée pour chacun des algorithmes lorsque l'objectif est de minimiser l'énergie sans impact négatif sur la performance. Dans ce cas là, autant pour First Fit que pour Best Fit, on remarque que trier les tâches par besoins en processeur décroissant est plus efficace que par besoins en mémoire. Du fait de la plus grande flexibilité de la ressource processeur (ressource

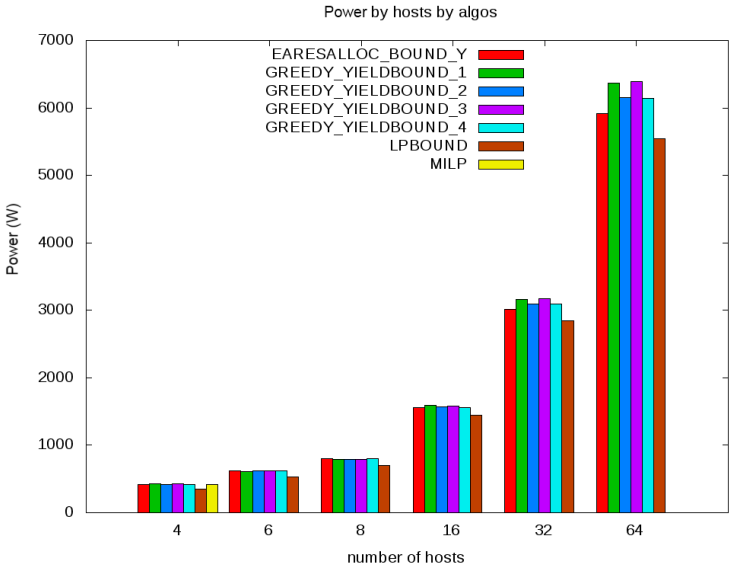


FIGURE 5.4 – Puissance consommée par les solutions résultantes de chaque algorithme, pour chaque taille d'instance. L'optimal (MILP) n'est calculé que pour des instances de 4 hôtes

fluide), il est plus facile de consolider sur un faible nombre de serveurs et donc de gagner en puissance.

Pour *EARESALLOC*, le fait de combler les déséquilibres permet une plus grande consolidation et donc d'éteindre encore plus de machines lorsque le système est composé d'un grand nombre de serveurs. L'avantage de cet algorithme est de pouvoir utiliser à la fois les deux listes.

La seconde étape consiste à maximiser la satisfaction des tâches avec un budget en watts borné. La figure 5.5 montre cette satisfaction dans le même cadre que la figure précédente. Cette borne est ici fixée arbitrairement à 60% de la consommation du centre de calcul. Dans ce cas là, trier par les besoins mémoire décroissant est plus intéressant. En effet, cette méthode va aboutir à un remplissage à 100% de la mémoire des hôtes. Comme les besoins mémoire et processeurs sont indépendants (suivant les hypothèses de [97]), elle mènera à une répartition globalement aléatoire des besoins en processeurs. Les tâches pourront alors quand même se relâcher et être moins sous pression pour cette ressource. Lorsqu'on consolide directement en triant sur le processeur, cette ressource sera directement très contrainte.

Dans les deux cas, la meilleure méthode consiste à ne pas faire un seul tri, mais à en faire plusieurs et à piocher en fonction des besoins dans la liste la plus pertinente. Les figures 5.4 et 5.5 montrent que *EARESALLOC* qui est basé sur ce principe est meilleur que les simples Best-Fit et First-Fit.

Pour une combinaison linéaire (de type $\alpha E_{norm} + (1-\alpha) Perf_{norm}$) des deux objectifs

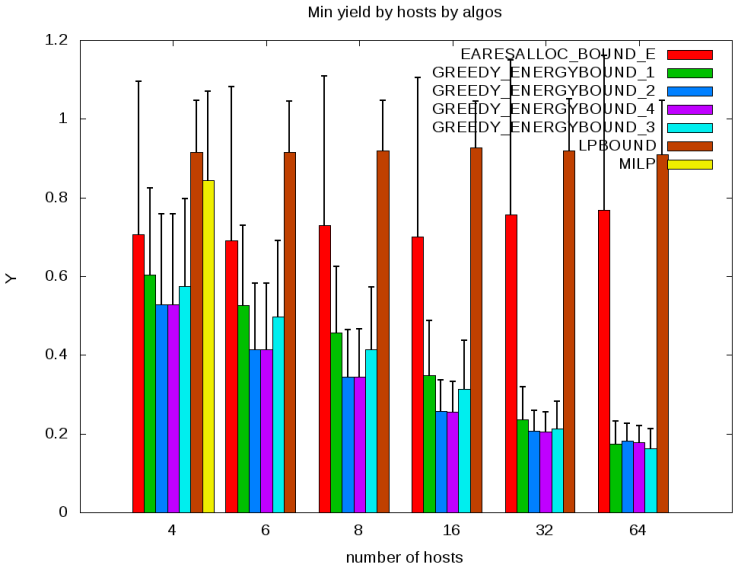


FIGURE 5.5 – Satisfaction minimum donné par les algorithmes en fonction du nombre d'hôtes dans un cas de puissance bornée.

normalisés, il est nécessaire d'étudier l'énergie (Figure 5.6) et la performance (Figure 5.7). Dans ce cas là, seul le First-Fit a été évalué car ses résultats y sont proches du Best-Fit. La métrique a été choisie avec un coefficient de 0.8, donc portant plus d'importance à la satisfaction des tâches qu'à l'énergie.

L'algorithme *GREEDY_MIXED_1* qui trie les tâches par besoin mémoire décroissant permet de consommer moins d'énergie. En effet il va consolider un maximum de tâches sur un minimum de serveurs, en ne tenant compte que de la contrainte rigide de mémoire. Il en découle que les tâches seront peu satisfaites car leur demande en processeur sera parfois seulement partiellement satisfaite. A l'opposé, *GREEDY_MIXED_2* va lui essayer de placer les tâches en réduisant l'impact sur la performance tout en satisfaisant aux contraintes mémoire. Lors du placement, une baisse de la satisfaction des tâches est acceptée (liée au coefficient de 0.8). Comme cet algorithme consolide sur un minimum de machine, il n'y a plus de capacité processeur à redistribuer aux tâches.

Enfin, l'algorithme *EARESALLOC* est meilleur et relativement proche de l'optimal pour la satisfaction et relativement mauvais pour l'énergie. En effet, en utilisant les deux listes, il lui est possible d'atteindre une satisfaction assez grande mais au prix de l'allumage de serveurs supplémentaires. Ces allumages ont un fort impact d'un point de vue puissance. Mais dans ce cas particulier, le ratio de 0.8 en faveur de la satisfaction fait que cette méthode est la meilleure au vu de la fonction objectif.

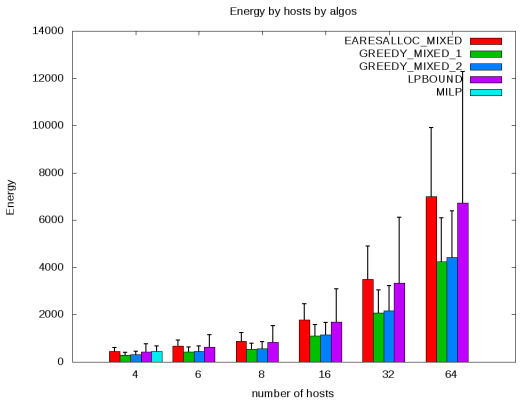


FIGURE 5.6 – Consommation de puissance résultante de l'allocation par les différents algorithmes pour chaque taille d'instance

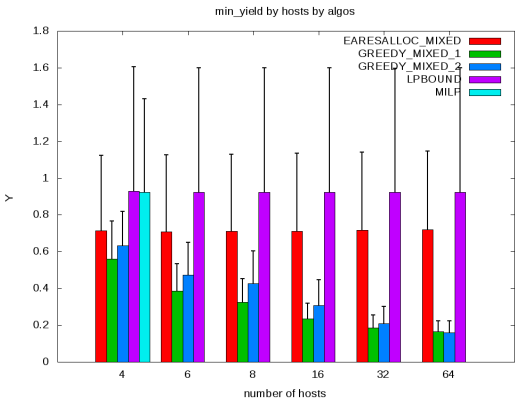


FIGURE 5.7 – Satisfaction minimum résultant de l'allocation par les différents algorithmes pour chaque taille d'instance

5.3.1.3 Conclusion sur les tris pour les algorithmes gloutons

Dans le cadre des multi-métriques et multi-contraintes, le tri occupe une place très importante. Un des éléments centraux est le fait que les contraintes sont fluides pour le processeur et rigide pour la mémoire. En effet, faire un placement avec comme contrainte principale la mémoire permet ensuite de manipuler plus aisément les allocations de processeur. Cet effet est visible sur la figure 5.5 où allouer à puissance bornée en triant sur les besoins mémoire permet ensuite d'avoir une meilleure satisfaction.

Enfin, dans tous les cas, être capable d'adapter le tri à la situation exacte (comme dans *EARESALLOC*) permet de mieux s'adapter à l'objectif que lorsque le tri est fait à priori.

5.3.2 De l'importance des métriques et du multi-objectif

Les méthodes d'ordonnancement classiques ont comme principal objectif la performance comme nous l'avons vu en section 2.3.1. Avec la montée en puissance des centres de calcul, le coté énergétique est devenu aussi une priorité. La majorité des études prend seulement en compte la performance, et la majorité du reste prend en compte la performance et l'énergie.

Il existe plusieurs méthodes pour prendre en compte ces multiples objectifs. La plus simple consiste à faire une combinaison linéaire d'objectifs normalisés. Ainsi on aura une formule du type $Metrique_{objectif} = \alpha \frac{Performance}{Performance_{max}} - (1-\alpha)\frac{Energie}{Energie_{max}}$. Le problème est alors concentré dans le choix de la valeur de α. Dans les études utilisant cette méthode il est souvent décrit l'effet de cette variation et c'est à l'utilisateur final de la choisir en fonction de son but.

Un problème plus fondamental est comment gérer de manière fine l'ensemble des

métriques intéressantes. En effet quel est l'impact sur la robustesse (voir Table 2.3) d'une optimisation de la performance ou de l'énergie ?

5.3.2.1 Multi-objectif par combinaison linéaire

Afin d'étudier ces impacts croisés entre métriques nous allons étudier un seul algorithme d'optimisation mais en modifiant sa fonction objectif.

Le concept général est simple, chaque métrique est normalisée (à valeurs entre 0 et 1) et un poids lui est assigné. Dans les exemples de la section précédente, il s'agit de la méthode *Mixed* appliquée à seulement la performance et à l'énergie.

Nous allons nous concentrer sur les quatre métriques suivantes :

Temps de réponse : La métrique de *Temps de Réponse* ou *makespan* d'un problème est le temps maximal nécessaire pour un placement particulier pour finir l'exécution de l'ensemble des machines virtuelles.

Énergie : La consommation énergétique totale est calculée en utilisant les modèles décrits en section 2.

Robustesse : La Robustesse d'un centre de calcul est lié à l'impact d'une défaillance (logicielle ou matérielle) sur un serveur. Il est modélisé par le nombre moyen de machines virtuelles allouées sur chaque serveur. Cette métrique doit être minimisée afin de maximiser la Robustesse.

Dynamisme : Le dynamisme est la capacité à répondre à une brusque augmentation de charge. On la modélise par la moyenne des capacités processeurs libres sur chacun des serveurs allumés.

Pour chacune de ces métriques, nous allons utiliser un algorithme génétique optimisant uniquement cette métrique. Nous allons aussi utiliser un algorithme optimisant de manière équilibrée l'ensemble de ces métriques.

GA_Energy optimise uniquement la métrique d'énergie

GA_RespT optimise uniquement la métrique de temps de réponse

GA_Rob optimise uniquement la métrique de robustesse

GA_Dyn optimise uniquement la métrique de dynamisme

GA_All optimise simultanément et équitablement ces quatre métriques.

La table 5.1 décrit les coefficients appliqués. L'objectif ici est de vérifier si prendre en compte une métrique dans l'objectif a un effet, aussi les pondérations choisies sont binaires : 0 ou 1.

Les figures 5.8, 5.9, 5.10 et 5.11 montrent en fonction de la charge l'évolution des différentes métriques (temps de réponse, énergie, robustesse, dynamisme) pour chacun des algorithmes.

Tout d'abord une remarque encourageante, l'algorithme lié à chaque métrique est le meilleur pour cette métrique. Ces graphes montrent aussi l'intérêt d'utiliser comme

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

TABLE 5.1 – Tableau récapitulatif des différentes versions de l'algorithme génétique associées à leurs coefficients d'optimisation de chacune des métriques.

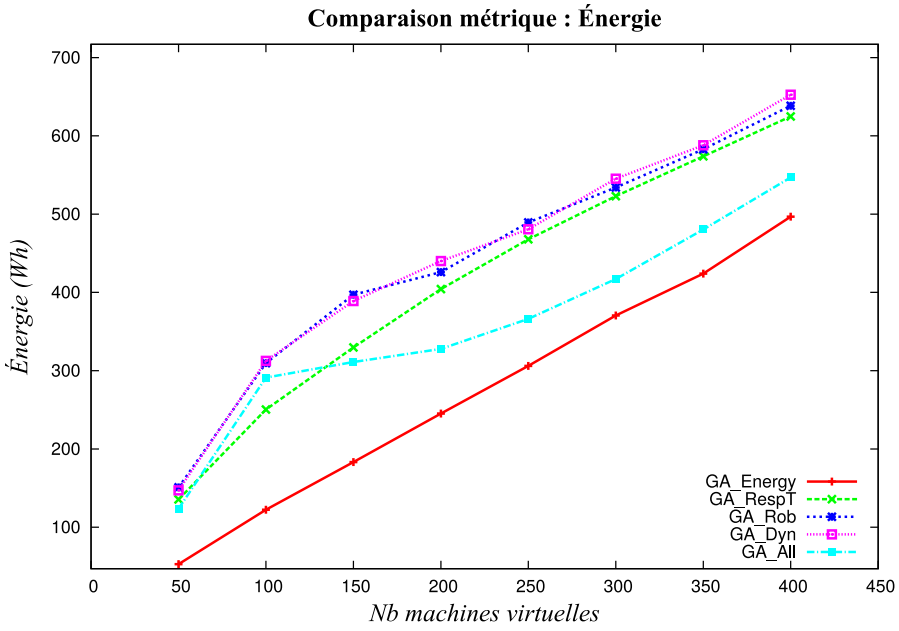


FIGURE 5.8 – Comparaison des résultats des différents algorithmes génétiques sur la métrique d'énergie. Les valeurs les plus petites sont les meilleures.

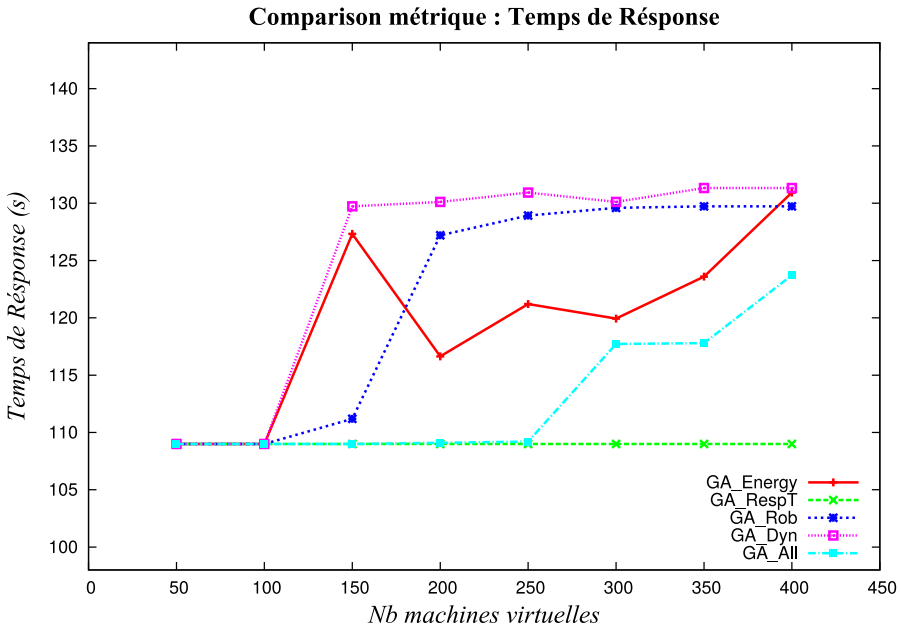


FIGURE 5.9 – Comparaison des résultats des différents algorithmes génétiques sur la métrique de temps de réponse. Les valeurs les plus petites sont les meilleures.

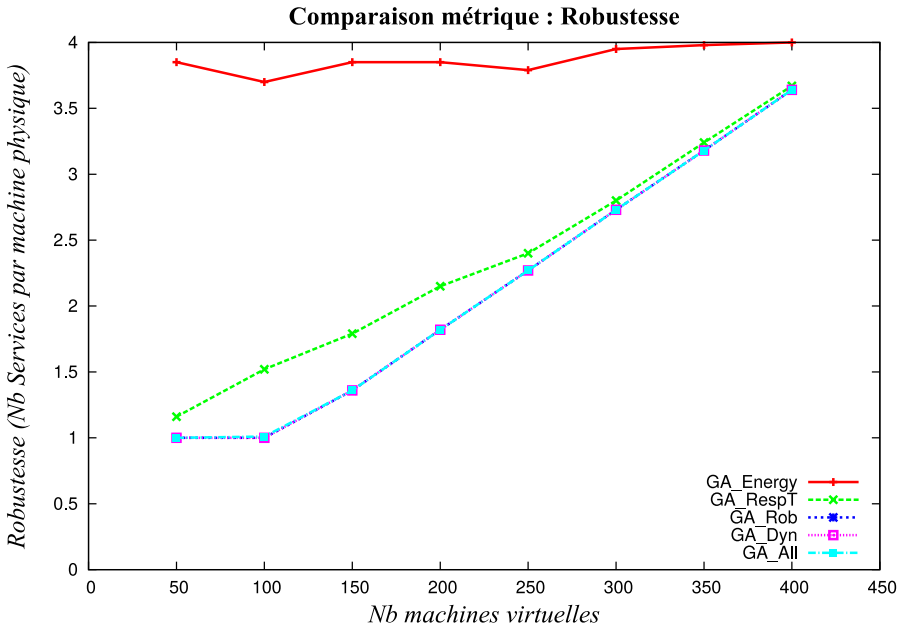


FIGURE 5.10 – Comparaison des résultats des différents algorithmes génétiques sur la métrique de robustesse. Les valeurs les plus petites sont les meilleures. Les courbes GA_Rob, GA_Dyn et GA_All sont confondues et de valeurs minimales.

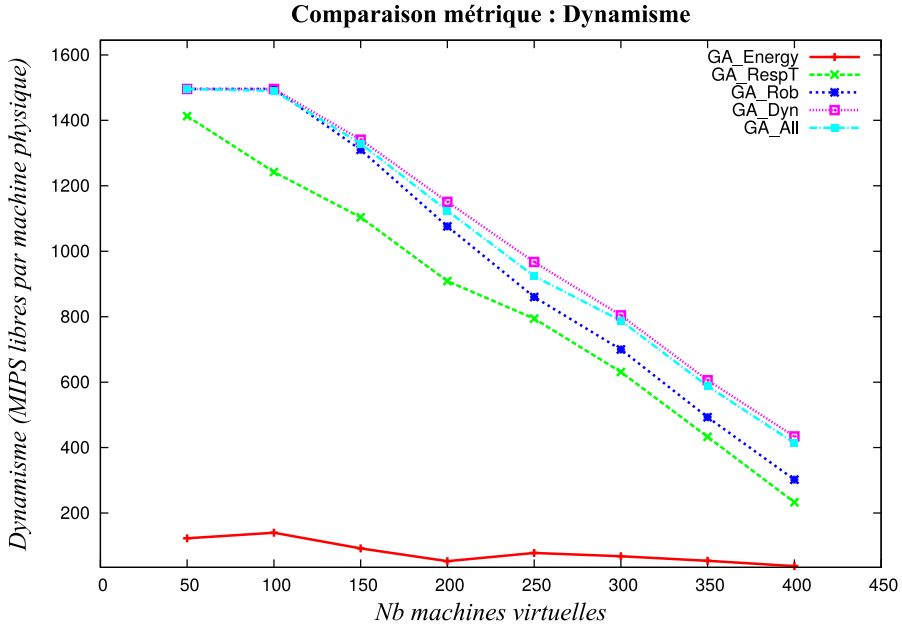


FIGURE 5.11 – Comparaison des résultats des différents algorithmes génétiques sur la métrique de dynamisme. Les valeurs les plus grandes sont les meilleures.

courbe de comparaison des algorithmes spécifiques à chaque métrique. Ainsi pour comparer un nouvel algorithme, il est intéressant de le comparer pour la métrique dynamisme à GA_Dyn, pour la robustesse à GA_Rob, etc.

Un second point intéressant est qu'aucun des algorithmes n'a un comportement identique. Il est ainsi impossible de considérer qu'optimiser pour une métrique particulière permettrait d'optimiser pour une autre. Il existe tout de même un certain nombre de synergies dans certains cas particuliers. Ainsi le fait que GA_RespT, GA_Rob et GA_Dyn finissent pour des raisons différentes à allumer un grand nombre de machines par rapport à GA_Energy qui lui tente de minimiser ce nombre fait que ces trois algorithmes ont un comportement de forme similaire.

Enfin, l'algorithme génétique qui optimise de manière équitable l'ensemble des métriques (GA_All) obtient des résultats généraux proches de l'optimal de chaque métrique.

En effet, lorsqu'on optimise pour une métrique on obtient souvent une famille de solutions dont la qualité est assez proche pour cette métrique. Le fait de tenir compte de plusieurs métriques permet alors de piocher une solution plus adaptée sans forcément avoir un impact négatif sur la métrique initiale. En résumé, le simple fait de tenir compte de plusieurs métriques permet d'être sûr que le comportement d'une métrique est maîtrisé.

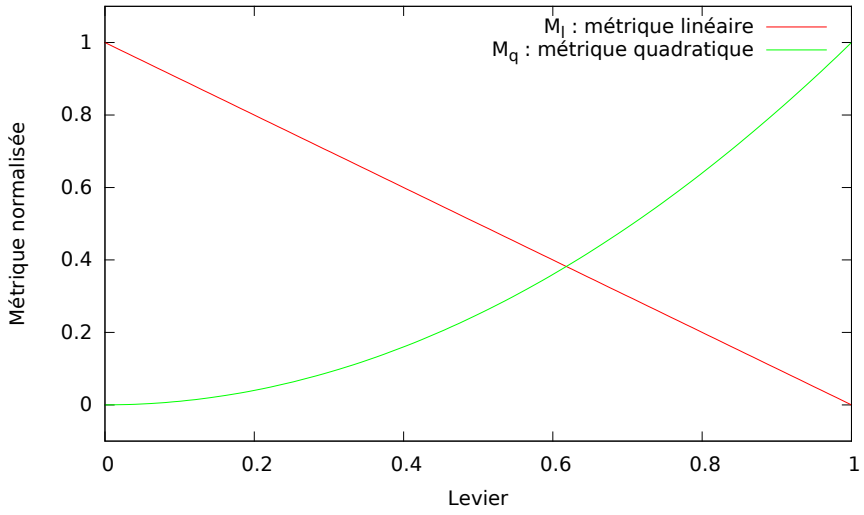


FIGURE 5.12 – Réaction d'une métrique linéaire et d'une métrique quadratique à l'utilisation d'un levier. Les valeurs basses de la métrique sont meilleures

5.3.2.2 Limites du multi-objectif par combinaison linéaire

La section précédente a montré qu'il est important de tenir compte de chaque métrique pour prendre une décision. Le problème principal vient du fait de choisir la fonction objectif. En effet, une simple formulation linéaire est problématique. Le premier problème concerne l'opérateur du centre de calcul qui doit être non seulement capable de paramétrer son système pour que la normalisation fonctionne, mais aussi capable d'exprimer ses besoins métiers sous la forme des α_i. Le second problème vient de la nature même des valeurs prises par les métriques.

En effet mixer dans la même équation une variable ayant un comportement linéaire et un comportement quadratique rend difficile le choix du α. Prenons par exemple un système ayant un degré de liberté (la fréquence d'un processeur par exemple) normalisé (ie. entre 0 et 1) appelé *levier*. Supposons avoir deux métriques, normalisées elles aussi, une quadratique M_q (telle que la consommation électrique) et une linéaire M_l (comme le temps de réponse) comme montré sur la figure 5.12 que l'on cherche à minimiser. De façon à avoir deux métriques antagonistes, on choisira par exemple $M_q = x^2$ et $M_l = 1 - x$.

Dans ce cas là, une formulation linéaire de la métrique à minimiser sera :

$$Obj_\alpha(x) = \alpha M_q + (1-\alpha)M_l = \alpha x^2 + (1-\alpha)(1-x) \tag{5.1}$$

La figure 5.13 montre cette équation pour différents valeurs de α. La courbe $\alpha = 0.5$ montre par exemple que si on choisit un $\alpha = 0.5$, la courbe sera optimale quand le levier est à la valeur intermédiaire (une fréquence intermédiaire dans notre exemple). Par contre on remarque que les comportements à $\alpha = 1/3$ et à $\alpha = 2/3$ sont différents.

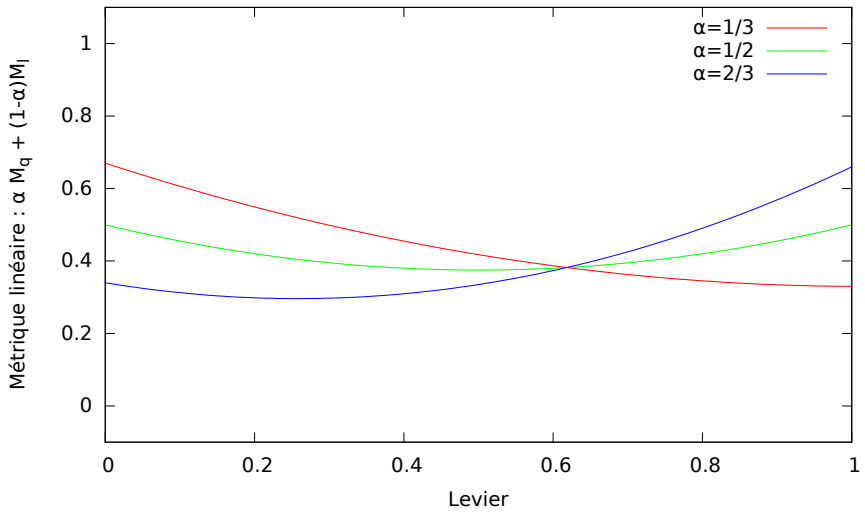


FIGURE 5.13 – Trois exemples de comportement de la métrique linéaire en fonction de l'utilisation du levier. M_q (resp. M_l) est la métrique quadratique (resp. linéaire)

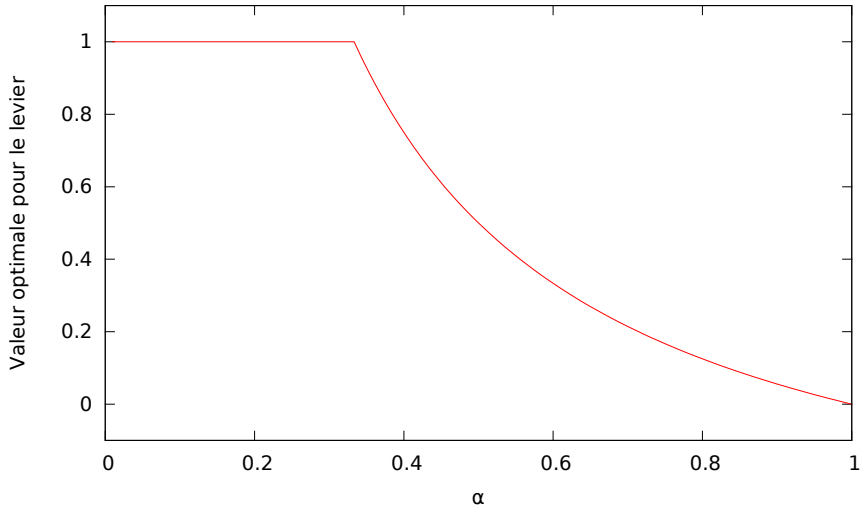


FIGURE 5.14 – Valeur optimale du levier en fonction de la valeur de α.

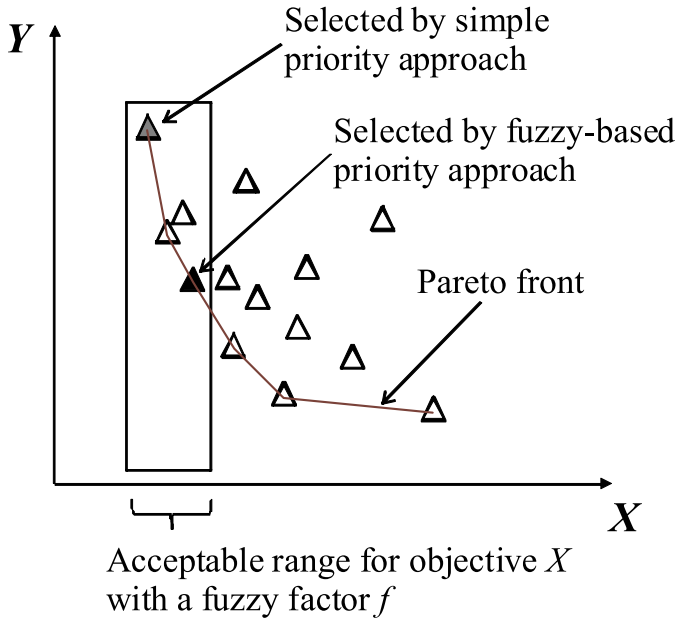


FIGURE 5.15 – Méthode de choix floue. On optimise en premier sur Y, puis après relâchement de $f\%$ on optimise sur X. On notera $\langle Y|X\rangle$

Lorsque $\alpha = 1/3$ la meilleure valeur du levier est 1 (fréquence maximale), on optimise donc le système comme le ferait la métrique linéaire M_l. Par contre lorsque $\alpha = 2/3$ on a un levier à 1/4. Ainsi on optimise à la fois les deux métriques.

La figure 5.14 montre l'évolution du meilleur levier en fonction de α. Elle décrit deux comportements très différents. Entre $\alpha = 0$ et $\alpha = 1/3$, le levier est fixé à 1 (fréquence maximale), et donc l'optimisation n'est pas réellement multi-objectif. Le comportement est tel qu'il serait si il n'y avait que la métrique M_l. A partir de $\alpha = 1/3$, le comportement est réellement multi-objectif mais encore compliqué à abstraire pour un utilisateur. De la même manière, lorsque α est à 1, le levier optimal est lui à 0 ce qui est cohérent car à ce moment là l'optimisation est semblable à ce qui se passerait avec seulement M_q.

Pour être capable de faire le choix de la bonne valeur de α il est donc nécessaire de connaître l'étendue des valeurs possibles afin de faire la normalisation, mais aussi leur distribution. Il n'est donc possible d'aboutir à un système équilibré que lorsqu'il est simple et donc linéaire, ou après une longue et complexe phase d'apprentissage.

5.3.2.3 Multi-objectif par algorithme flou

Nous avons proposé[J9] une autre méthode qui consiste, comme le montre la figure 5.15, à faire plusieurs pas, un par métrique. Dans cette méthode on suit le principe décrit dans l'algorithme 2 et l'on notera $\langle Metrique_1|Metrique_2|...|Metrique_n\rangle$ dans la suite.

L'intérêt de cette méthode par rapport aux méthodes linéaires est d'être beaucoup

Algorithm 2 Algorithme flou

1: **for all** placement de tâches (appelés *position*) **do**
2: Calculer toutes les métriques
3: **end for**
4: **for all** métriques sauf la dernière **do**
5: Calculer la valeur optimale
6: Garder seulement les *positions* à moins de $f\%$ de l'optimal
7: **end for**
8: Sélectionner la meilleure position pour la dernière métrique

plus simple pour l'utilisateur final. Il n'est pas nécessaire de normaliser les métriques et donc d'en avoir une modélisation pour un cas particulier. De plus le choix de f, même si il reste potentiellement difficile, est plus simple à appréhender. f est le pourcentage de perte par rapport à l'optimal que l'on est prêt à accepter. Un dernier avantage est que cette méthode garantie de rester sur le front de Pareto. En effet elle ne sera pas dominée par une autre solution dans toutes les autres métriques car les autres solutions sont soit inférieures pour les métriques de 1 à $n-1$, soit sont inférieures pour la métrique n.

Une fois cette méthodologie posée, elle est utilisable de plusieurs façons. Soit en testant effectivement tous les placements possibles, soit en utilisant un sous ensemble pertinent grâce à des heuristiques. La première méthode a l'avantage de l'exhaustivité mais présente le défaut d'avoir un fort coût en calculs et donc d'être irréaliste (voir Section 4.1).

La seconde méthode a l'avantage de ne considérer que les points pertinents du front de Pareto. En effet, sur l'ensemble des placements, une grande majorité est médiocre relativement à l'ensemble des métriques. Les points ainsi créés seraient loin de ce front, et baisseraient la qualité du placement. Il est donc nécessaire d'avoir un ensemble d'heuristiques de placements pertinentes.

A valeur d'exemple nous allons utiliser les algorithmes gloutons suivant dont le comportement en fonction de la charge est montré en figure 5.16 :

Uniform Les tâches sont assignées aléatoirement sur les serveurs possédant assez de ressources ;

MinHR Les tâches sont assignées en priorité sur les serveurs contribuant le moins à la recirculation de la chaleur (*D-matrix* en section 2.1.2.2) ;

CoolestInlet Les tâches sont assignées sur les serveurs dont la température d'arrivée d'air est minimale ;

Perf-Aware Les tâches sont assignées sur les serveurs où leur temps d'exécution sera le plus court ;

Energy-Aware Les tâches sont assignées sur le serveur où l'impact en consommation énergétique (calcul et refroidissement) est minimal ;

Thermal-Aware Les tâches sont assignées sur le serveur où l'impact sur la température d'entrée des serveurs sera minimale.

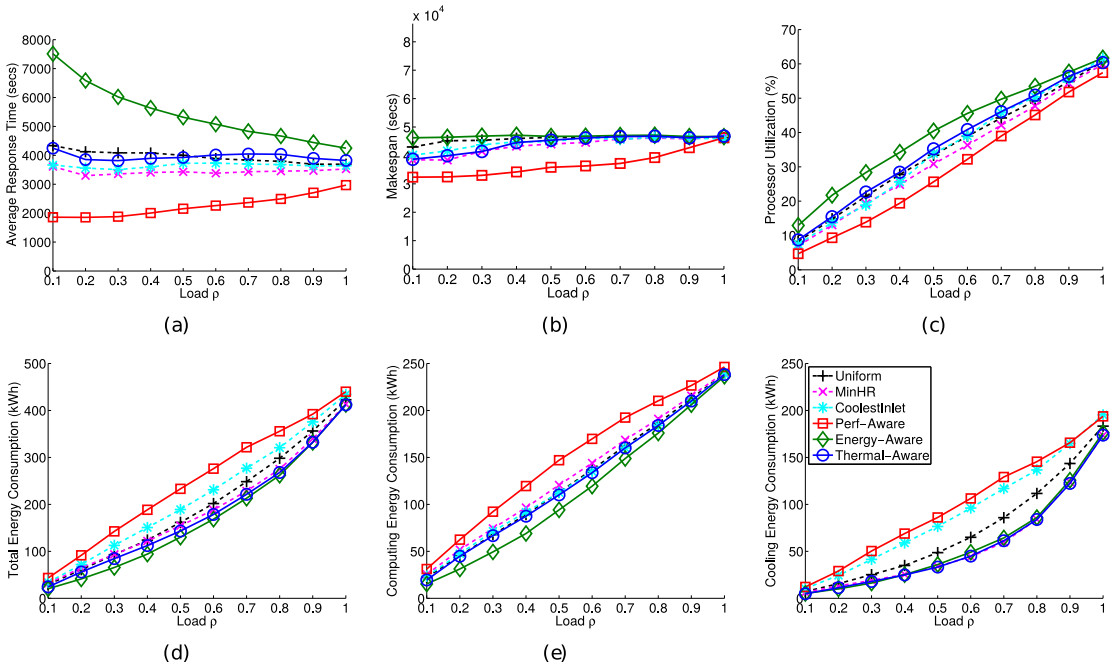


FIGURE 5.16 – Valeurs des métriques pour les six heuristiques mono-objectifs sur des processeurs x86 de performances diverses. La légende s'applique à toutes les sous-figures.

Une fois de plus remarquons l'impact important du tri (ici des machines) sur les différentes performances des algorithmes gloutons.

Pour un cas de charge moyenne, le comportement de la méthode de choix floue pour $\langle Energy|Performance\rangle$ est montrée sur la figure 5.17. La Figure 5.18 montre plus précisément les valeurs des métriques d'énergie et de performance en fonction du facteur de flou (*fuzzy factor*).

Sur cet exemple particulier la charge du système est haute (load $\rho = 0.8$). Les résultats sont semblables dans les autres cas et une étude complète est disponible dans [J9]. La figure 5.18 montre qu'avec l'augmentation de f on améliore la performance. C'est cohérent car on relâche la contrainte *énergie minimale* ce qui laisse de la liberté pour optimiser la performance. Pour une perte minime en énergie (jusqu'à $f = 0.9$) on peut sélectionner des solutions qui améliorent grandement la performance.

Enfin, il est possible de mixer la méthode floue et la méthode linéaire par exemple en utilisant $\langle Energie|Performance|\alpha Resiliance + (1-\alpha)Dynamisme\rangle$

Par rapport à une méthode de combinaison linéaire, la méthode floue a trois principales différences :

L'ordre Dans la méthode floue, l'ordre des métriques utilisées est prépondérante. En effet, la principale garantie de qualité obtenue avec cette méthode est sur la première métrique. C'est de moins en moins vrai pour les autres métriques.

Pas de connaissances à priori La méthode floue résout le problème de la métrique mal normalisée ou à comportement non-linéaire qui est décrit dans la section précédente.

Utilisation externe à des algorithmes La méthode floue ne peut pas être utilisée facilement à l'intérieur d'un algorithme pour choisir dans une étape interne entre deux leviers possibles, ce que permet la méthode linéaire. Les deux par contre permettent de choisir la *meilleure* solution parmi un grand nombre de placements possibles et sont donc utilisables dans le cas des algorithmes génétiques par exemple.

Pour conclure, les deux méthodes sont complémentaires, la méthode linéaire est plus simple et plus générique mais nécessite une grande rigueur de mise en place pour être pleinement utile. La méthode floue est particulièrement adaptée lorsqu'on a une métrique principale mais que l'on veut quand même tenir compte d'autres métriques secondaires.

5.3.3 La prise en compte du temporel

La prise en compte du temporel est complexe car multi-niveau. D'un coté l'inertie thermique est un phénomène continu, de l'autre les latences d'application de leviers (allumage d'un serveur par exemple) sont discrètes. De par leur nature différente, ces deux effets temporels que sont l'inertie thermique et les évènements qui remettent en question la situation actuelle doivent être traités différemment. Nous avons abordé en section 3.3.1 les différentes formes de décisions. La suite de cette section va illustrer les particularités des décisions liées aux problématiques thermiques et à celles de de la gestion de l'état des machines.

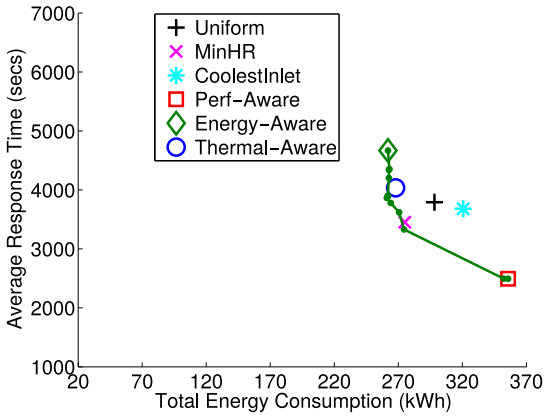


FIGURE 5.17 – Le trait vert parcours les solutions sur le front de Pareto lorsque f passe de 0 (en haut à gauche) à 1 (en bas à droite) pour un cas où la charge est haute.

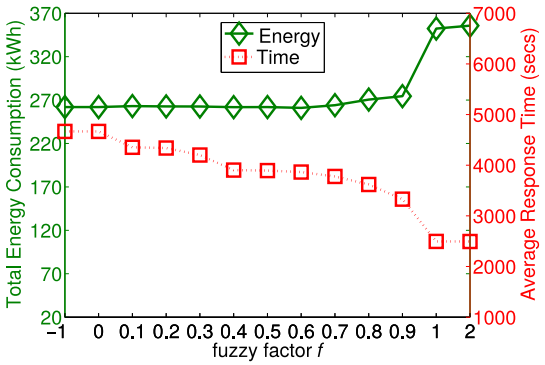


FIGURE 5.18 – Pour le même exemple qu'en Figure 5.17, on représente en fonction de f les valeurs des métriques pour un cas où la charge est haute

5.3.3.1 Thermique

Dans [C17, J9] nous avons étudié l'impact thermique sur état stable, lorsque la température est stabilisée. Il est possible d'aller plus loin et de prendre en compte la montée en température. Ainsi même sans événement extérieur au système il est possible d'avoir une évolution au cours du temps et donc de ne pas avoir d'état d'équilibre. En effet, intrinsèquement, la température évolue de manière lente du fait de l'inertie thermique du matériel.

Le simple fait d'avoir des ventilateurs empêchent d'avoir un état stable. La figure 5.19 montre l'évolution de la température et de la consommation électrique dans une simulation simple où un serveur exécute une tâche consommant 100% de ses ressources processeur.

Alors que la consommation de ressources est constante dans la première partie (avant 1666s), la consommation électrique varie du fait du déclenchement des ventilateurs. On peut remarquer que sur une expérience similaire (Figure 5.20) réalisée sur Grid'5000 que le comportement est semblable et qu'il est possible de dénombrer les 4 allumages de ventilateurs.

Les ventilateurs sont asservis en température comme décrit en section 2.1.2.2. D'une manière générale, les systèmes de refroidissement, ventilateurs, mais aussi les systèmes de refroidissement au niveau des serveurs et des baies de serveurs, fonctionnent selon se principe. Ainsi plus le niveau de température est haut, plus la consommation électrique le sera aussi. La consommation électrique augmentera ainsi de manière sur-linéaire par rapport à la température, d'où l'objectif de minimiser la température maximale. Il est à remarquer que cette déperdition est dans le même sens que celle concernant les courants de fuites du fait de la technologie CMOS décrite en section 2.3.1.

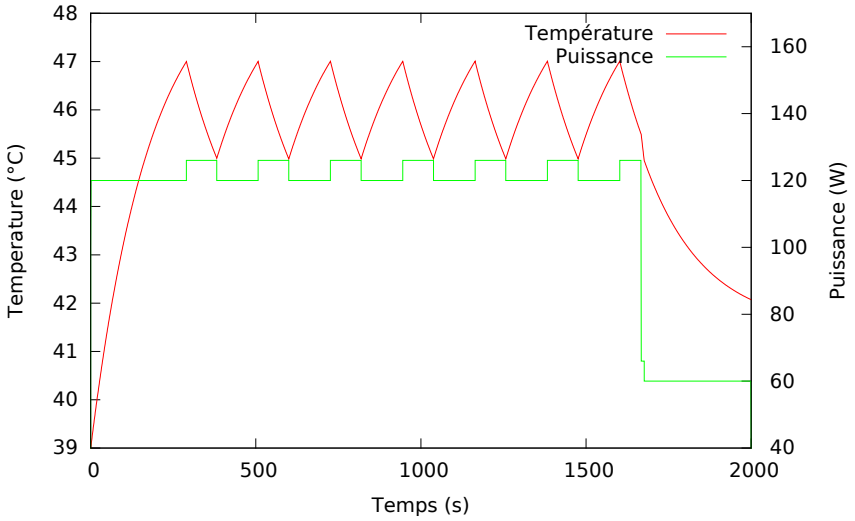


FIGURE 5.19 – Simulation de la température d'un processeur (de type Intel Xeon E5-2630). Elle est liée aux jobs présents (ici 100% jusqu'à 1666s puis 25% après) et aux ventilateurs qui se déclenchent à partir de $47°C$

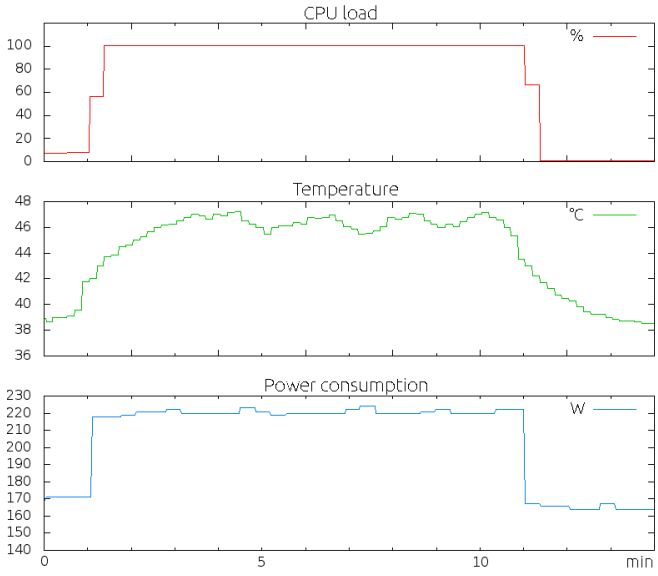


FIGURE 5.20 – Évolution de la charge, température et consommation de puissance mesurées sur un serveur de Grid'5000 avec une charge simple

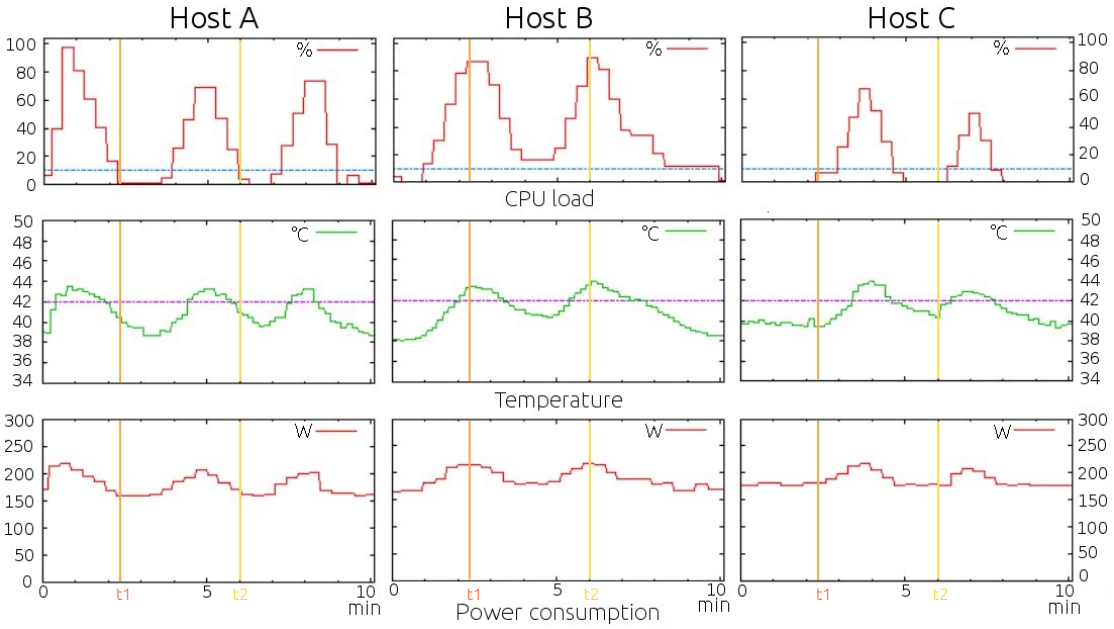


FIGURE 5.21 – 6 machines virtuelles démarrent à $t = 0$ sur la machine A. Un système événementiel autonome les migre pour éviter le démarrage des ventilateurs.

Du fait des latences de montée en température, pour optimiser au maximum l'énergie totale consommée par un centre de calcul, il n'est donc pas possible de simplement faire un placement initial très efficace.

Il est donc nécessaire de prendre en compte l'évolution temporelle de la température. Il existe deux moyens : Évènement seuils de température ; Pré-calcul du temps nécessaire à atteindre une certaine température.

Un exemple d'utilisation sur Grid'5000 est donné dans [C26]. Nous y décrivons un système autonome réactif qui tente de garder la température de chaque serveur en dessous d'un seuil en migrant les tâches présentes sur les serveurs les plus chauds vers les serveurs les plus froids. Un exemple du dynamisme de ce système est visible sur la figure 5.21.

On remarque que lorsque des machines sont allumées il est plus intéressant d'avoir un système de ping-pong entre plusieurs serveurs. Une position stable des VMs entraînerait le démarrage des ventilateurs. Il est alors nécessaire de comparer le surcoût de la migration et celui de l'évacuation de la chaleur.

Cette méthode permet de plus de rationaliser l'utilisation des serveurs de secours, serveurs supplémentaires souvent allumés même à vide pour permettre d'absorber rapidement des pics de demandes de ressources. Au contraire, lorsque la charge est très stable et prédictible, le sur-coût de l'allumage d'un serveur supplémentaire n'absorbe pas l'économie d'un ventilateur.

En plus de cette économie d'énergie, une gestion fine de la température par migration ou par ralentissement de la fréquence des serveurs permet de réduire la température maxi-

male atteinte par le matériel. Dans [J12] nous avons étudié par simulation la différence entre des politiques classiques et des politiques utilisant entre autre des changements de fréquences pour réduire la dissipation thermique et donc éviter l'utilisation de ventilateurs. Nous avons ainsi pu réduire la consommation électrique globale (IT+cooling) de plusieurs pourcent tout en réduisant les températures maximum et moyenne atteintes par les serveurs. Réduire ces températures permet de d'améliorer la durée de vie du matériel[82].

5.3.3.2 Condition d'utilisation de la prédiction

Dans les sections précédentes le focus a été principalement mis sur l'allocation des tâches sur les serveurs. Il s'agit alors de trouver le meilleur placement à un instant donné. Pourtant lorsque le temporel est pris en compte il n'est pas possible de considérer uniquement une suite de tels placements. Cette approximation n'est possible que lorsque le dynamisme des tâches est très faible par rapport aux sur-coûts du dynamisme du système. Pour des tâches au profil constant pendant plusieurs heures, l'allumage d'une machine peut être considéré comme négligeable.

Lorsque les tâches ont un dynamisme du même ordre de grandeur que celui du système ou plus grand, il est nécessaire de faire des compromis.

Par exemple, dans un cas réel dont les traces des serveurs web sont librement accessibles, celui de la coupe du monde de football 1998 en France[66], le nombre de requêtes par seconde peut changer d'un ratio 50 d'une seconde à l'autre. Sur des échelles de temps plus raisonnables de quelques minutes, c'est toujours vrai, même si ce ratio passe à 5. Ce cas réel, illustré sur la figure 5.22, montre que dans certains cas les constantes de temps de modification du système (allumage ou extinction de serveurs) sont largement supérieures à celles liées au dynamisme des tâches.

Dans [C4] et [J11] nous avons montré comment trouver, pour chacune des secondes, la combinaison de serveurs la plus efficace pour le cas de la figure 5.22. L'objectif est de se rapprocher d'un système proportionnel tel que décrit au début du chapitre 3. Les gains sont substantiels car sur la charge de la coupe du monde 1998, un système proportionnel permettrai d'atteindre une économie d'énergie de 35% par rapport à des serveurs *classiques* non-proportionnels[C4], et avec une combinaison de serveurs de différents types (RaspberryPi, Atom, I7) il est possible de se rapprocher de cette proportionnalité, mais seulement avec un gain de 8% et non 35% (voir Figure 5.23).

Par contre, il s'agit d'une étude où les coûts (migration, allumage, extinction) sont nuls. Dans [J11], avec une combinaison différente (ARM Cortex-A15, Intel Xeon, AMD Opteron) sur le même jeu de données, nous avons montré qu'il est possible d'avoir un gain de 17% par rapport à un centre de calcul homogène. Dans ces deux études, l'hypothèse est faite d'avoir un oracle parfait. De plus dans la seconde, nous avons étudiés l'impact du coût d'allumage et d'extinction. Pour une valeur relativement haute (30 à 300 joules), il devient même plus intéressant de n'utiliser que des machines très puissantes car la meilleure chose à faire est de ne jamais ni allumer ni éteindre des machines.

Ainsi deux éléments sont indispensables lorsque le temps est rajouté :

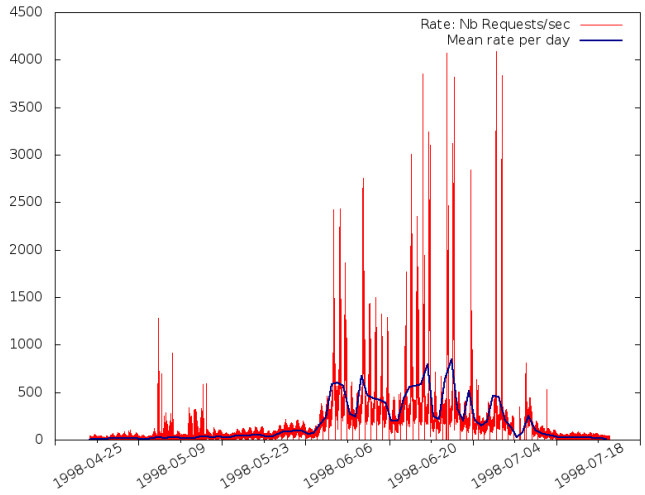


FIGURE 5.22 – Haute variabilité des traces d'accès au serveurs web de la coupe du monde de football 1998.

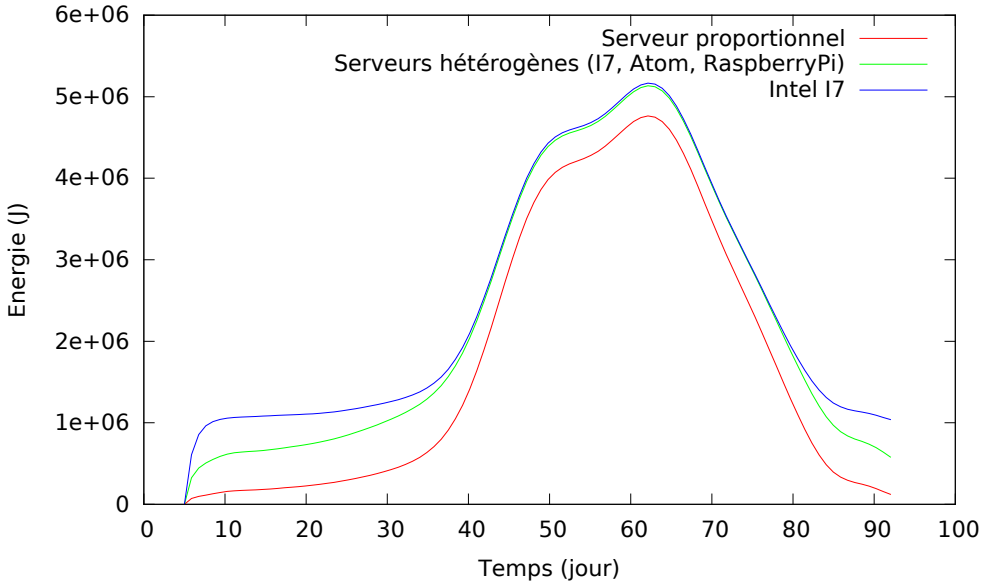


FIGURE 5.23 – Puissance nécessaire pour différent types de centres de calcul pour fournir les pages web de la coupe du monde 1998

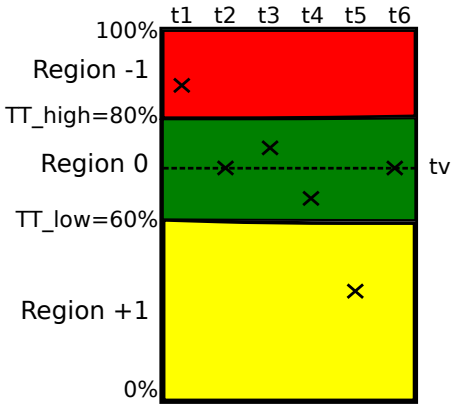


FIGURE 5.24 – Principe de la gestion de charge par hystérésis. Si un serveur est chargé à plus de 80% ou à moins de 60%, le système autonome réagi

— La gestion de la qualité de service. Contrairement au problème du placement, il peut être intéressant de ne pas gérer un pic de demande de ressource si il est ponctuel. Encore une fois il y aura un compromis à trouver entre qualité de service et efficacité énergétique ;
— Un oracle dont la qualité de précision impactera forcément les gains d'énergie et la qualité de service. Il sera alors important d'évaluer l'impact de cette prédiction sur les métriques.

L'impact de ces deux éléments sera lié à l'ordre de grandeur du dynamisme des données par rapport à l'ordre de grandeur des temps de réaction du matériel.

5.3.3.3 Gestion de l'allumage/extinction des serveurs

Dans le cas où l'évolution des demandes en ressources est moins rapide que l'échelle de temps de l'allumage et l'extinction de serveurs et la migration de tâches, il est possible d'utiliser la prédiction pour réduire au maximum les défauts de qualité de service sans perdre de vue l'énergie.

Comme décrit dans la partie précédente, lorsque l'objectif est une combinaison entre énergie et performance, il est nécessaire d'observer une certaine inertie dans les décisions. Ainsi un fort pic très court lors d'une longue phase constante ne provoquera pas forcément un allumage de serveurs.

Il faut donc séparer deux éléments :

La prédiction qui va fournir le profil de charge dans le futur ;

La politique de gestion de serveurs qui va décider en fonction de la prédiction l'allumage et l'extinction de serveurs.

Pour expliciter cette différence, nous avons étudié[C2] la méthode classique consistant à utiliser un algorithme à hystérésis (semblable à l'algorithme du gouverneur de DVFS Conservative décrit en section 3.2.2) décrit en Figure 5.24.

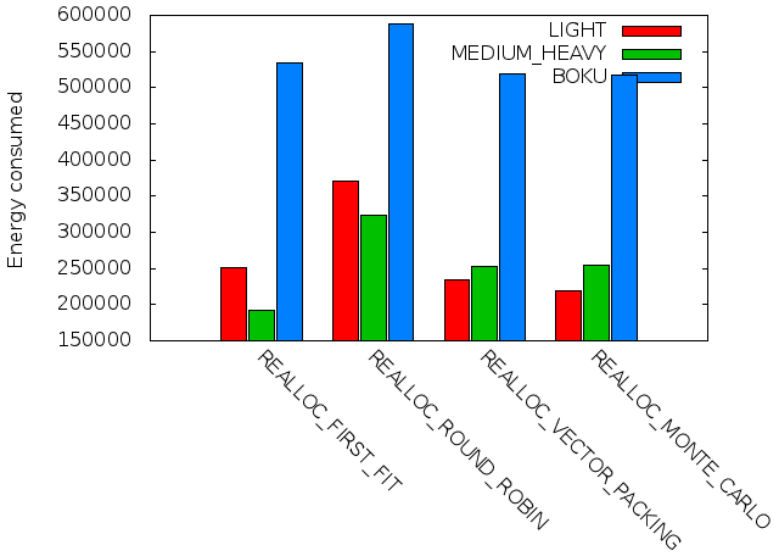


FIGURE 5.25 – Consommation énergétique pour les algorithmes sur deux types de charges, faible et moyenne/haute.

Dans ce cas là, la prédiction est que si la charge d'un serveur est supérieure à 80% elle va augmenter, mais que si elle est inférieure à 60%, elle va diminuer. L'algorithme de décision utilisant l'information est que si pour un serveur un tel comportement est détecté, il va falloir allumer ou éteindre exactement un serveur.

D'autres algorithmes sont utilisables tels que les réseaux de neurones ou la régression linéaire par exemple pour la prédiction. Une politique de gestion de machines classique est de laisser toujours autant de serveurs allumés que de serveurs utilisés (technique des *2N*).

Ces oracles et politiques sont intimement liés aux algorithmes de placement et de re-placement associés.

Les figures 5.25 et 5.26 montrent la consommation énergétique et les violations de qualité de service associés au système à hystérésis (décrit en Figure 5.24). Il y a violation de qualité de service lorsque les ressources allouées sont inférieures aux ressources nécessaires. Nous avons initié l'exploration de ces systèmes lors de la visite à l'IRIT durant deux semaines de Michael Maurer, doctorant de l'Université de Vienne de l'équipe d'Ivona Brandic en mars 2011.

Ces deux figures utilisent trois profils de charges dans un système où il y a une remise en cause régulière de l'allocation des machines virtuelles :

Boku est une charge stable dans le temps mais consommatrice de beaucoup de ressources. Sur la Figure 5.26, Boku n'a aucune violation de qualité de service quelque soit la méthode car il n'y a aucun changement dans le temps.

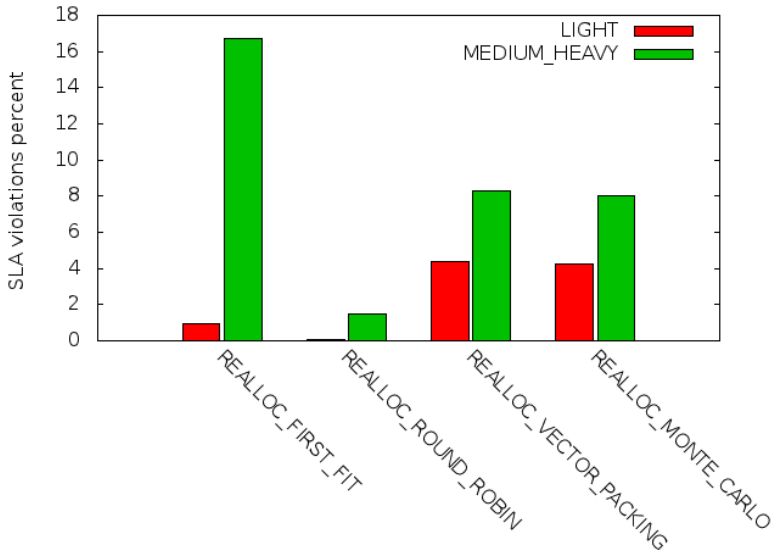


FIGURE 5.26 – Violations des demandes de qualité de service pour les expériences de la Figure 5.25. Boku étant très stable il n'y a aucune violation pour ce profil de charge.

Medium_Heavy a une volatilité forte (jusqu'à 50% de changements entre deux prises de décisions)

Light a une volatilité faible (10% de changements par pas de temps)

Dans un premier temps il faut rappeler l'importance de l'heuristique de placement initial dans le cas des charges évoluant peu (profil Boku). Pour un profil stable, le placement initial est déterminant de la qualité (ici énergétique) de l'ensemble de l'allocation.

Concernant les profils de charges plus variables, deux catégories sont visibles ici. Les algorithmes à base de consolidation (FirstFit, VectorPacking, MonteCarlo) et à base de répartition (RoundRobin). Ce dernier aura tendance à allumer plus de machines mais aussi à provoquer moins d'allumages/extinctions et à éviter les violations de qualité de service. En effet, avec cette heuristique, une majorité des serveurs aura comme charge le minimum, laissant de la place pour les variations à la hausse.

Les algorithmes par consolidation vont avoir tendance avoir des serveurs chargés proches de leur capacité maximale. Ils vont donc devoir allumer et éteindre des machines plus souvent. Ils auront aussi une capacité moindre à absorber des pics de charge.

Le choix de l'heuristique se fera donc en fonction des rapports entre la variabilité de la charge, les latences de re-configuration des serveurs, ainsi que de la qualité de service visée. Si le dynamisme (Section 5.3.2.1) n'est pas un des éléments de cette qualité de service, la gestion fine des ressources disponibles sera elle aussi moins primordiale.

5.4 Conclusion

Malgré le fait d'être étudié depuis le début des systèmes distribués, la problématique de la pertinence des heuristiques est toujours prégnante. En effet, avec l'augmentation de la complexité des systèmes, de leurs possibilités mais aussi de ce qui en est attendu, les heuristiques doivent manipuler une quantité de connaissances de plus en plus large. Alors que dans un système simple il est facile de produire des heuristiques gloutonnes efficaces, dans un tel système avec des objectifs variés, les méthodes par raffinements (algorithme génétique) sont beaucoup plus souples.

Pour les algorithmes gloutons, l'élément le plus important consiste à choisir le tri en fonction des objectifs. Ce choix a alors un impact fort avant de commencer l'algorithme. Pour l'algorithme de Vector Packing, le tri est remis en cause après chaque choix, apportant une meilleure adéquation avec un objectif composé.

Lorsque l'objectif est composé, il devient difficile d'évaluer les choix individuels, mais aussi le choix final. Deux niveaux sont disponibles lors de l'utilisation du multi-objectif : Tenir compte de plusieurs objectifs, être capable de prioritiser les objectifs. Pour le premier niveau, une simple combinaison linéaire permet de gérer plusieurs objectifs, comme nous l'avons vu avec les algorithmes génétiques. Par contre, lorsqu'il s'agit de prioritiser il devient nécessaire d'utiliser des méthodes plus poussées, telles que l'utilisation de méthodes floues.

Une partie de la complexité de la modélisation précise des centres de calcul vient de la prise en compte de la température, avec ses effets d'inertie. Il devient pertinent d'avoir des effets de ping-pong qui sont meilleurs que des états stables, ce qui est contraire à l'usage du domaine.

Enfin, l'inertie vient aussi des problématique d'extinction et d'allumage des serveurs, et de la comparaison entre ces constantes de temps et celles liées au dynamisme des applications. Les contraintes sur la prédiction de charge seront très différentes en fonction de ce ratio, mais aussi des objectifs de qualité de service.

Chapitre 6

Vers une vision distribuée

> I would like to see anyone, prophet, king or God, convince a thousand cats to do the same thing at the same time
>
> Neil Gaiman, Sandman #18
> "A Dream of a Thousand Cats"

Plusieurs éléments montrent qu'on se dirige vers une multiplication des centres de calcul. La forte croissance des besoins en calculs mène à l'implantation de nouveaux centres de calcul de grande taille. De plus le développement des systèmes de type proportionnel va mener à la multiplication des *petits* centres de calcul appelés *Data Center in the Box* dans les prochaines années. En effet de par les propriétés de proportionnalité le besoin de forte densité sera de plus en plus réduit. La densité sera même vue comme une contrainte du fait de l'obligation de gérer la chaleur produite. Cette vision à long terme rend nécessaire d'étudier l'applicabilité des méthodes classiques des systèmes distribués au cas du Cloud afin d'ouvrir la porte à de futures études mais aussi de montrer que cette direction est pertinente d'un point de vue technique et industriel.

Les techniques et outils décrits précédemment posent un certain nombre de problèmes classiques[98] de par leur nature centralisée : problème de latence, de sur-coût de maintient d'une vue globale, de gestion centralisée des droits. Il s'agit là des différents challenges qui ont servi de prélude à la création des grilles[41] de calcul.

La réponse actuellement apportée dans les systèmes de type Cloud est différente. La vision multi-institution coopérative est bien moins présente que dans la communauté grille de calcul. Le paysage consiste plus en un petit nombre de très grands acteurs possédant plusieurs centres de calcul, et d'un grand nombre d'acteurs moyens indépendants. On se retrouve donc à travailler soit totalement en interne, soit à déléguer en externe en achetant à un unique acteur. Cette simplicité (virtualisation) des interactions rend la gestion de ressources externes distribuées plus simple que dans le domaine des grilles de calcul.

Autant en interne (propriétaire de plusieurs sites) qu'en externe les possibilités majoritairement utilisées sont :

— La possibilité de soumettre des tâches à un à deux sites ;
— Les sites gèrent les tâches en interne sans visibilité extérieure.

Il y aura donc deux types de décisions à prendre : des décisions à impact global distribuées ; des décisions locales à impact local.

6.1 Décisions à impact global distribuées

Il s'agit ici du cas où une structure veut utiliser des ressources sans avoir à se préoccuper de savoir si elle sont locales ou distances. Il s'agit donc du cas où l'on a des ressources locales et accès à des ressources distantes (*cloud hybride*) ou du cas où l'on a uniquement accès à des ressources distantes (*cloud* "classique"). Dans le cas général où on ne se limite pas à un cloud privé et un cloud externe mais où il y en a potentiellement un grand nombre, on parle de *fédération* de clouds.

Le problème est alors le suivant : Étant donné une tâche, sur quel site (local ou distant) l'exécuter sans connaissance globale du système.

Pour résoudre ce problème, les techniques du monde du Pair à Pair[71] sont particulièrement adaptées. Elles présentent en effet souvent de bonnes propriétés : pas de point de défaillance, pas de besoin de vue globale, pas de système centralisé d'autorité à mettre en place.

6.1.1 Ordonnancement coopératif

Une grande majorité des travaux précédent avaient une vision statique[53, 77] et centralisée[51, 93]. Ils se basent sur la présence d'un serveur ayant une vision globale et prenant toutes les décisions sur l'ensemble du système et une fois la décision prise du centre de calcul où est allouée une VM, la décision n'est plus remise en question.

Des méthodes hiérarchiques[38, 109] ont aussi été proposées pour résoudre ce problème, mais elles sont toujours limitées par la nécessité d'avoir une seule infrastructure de décision.

Une réponse préliminaire est proposée par DVMS[83] (*Distributed VM Scheduler*) qui propose une approche statique et coopérative de la gestion de machines virtuelles dans un ensemble de centres de calcul. De manière générale, les approches distribuées de gestion de VMs fonctionnent sur le principe d'objectif commun, ce qui est peu applicable lorsqu'une fédération est constituée d'entités indépendantes.

De manière générale et comme pour un ordonnancement centralisé, pour un ordonnancement coopératif, deux décisions sont à prendre : le placement initial ; le replacement régulier ou sur événements.

Nous avons proposé[C19] un algorithme coopératif et dynamique de gestion de centres de calcul. D'un point de vue structure, on peut le rapprocher d'un algorithme de type Super-Peers[10].

La figure 6.1 montre cette structure coopérative entre les centres de calcul. Un gestionnaire peut être associé à un centre de calcul local ou non. Si il l'est, on est dans un

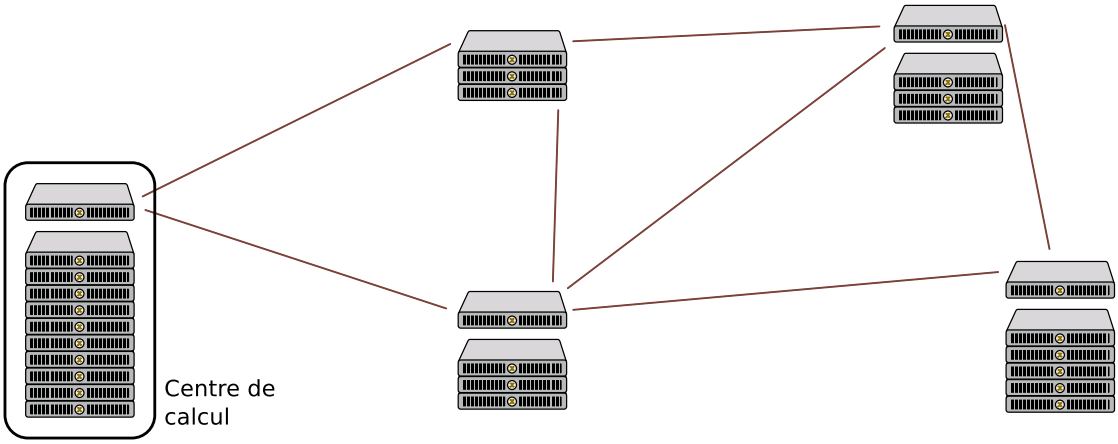


FIGURE 6.1 – Pour un système coopératif de centres de calcul, des gestionnaires locaux coopèrent pour la gestion des tâches

cas de cloud hybride, avec des ressources locales et la possibilité d'utiliser des ressources distantes.

6.1.1.1 Soumission coopérative de tâches

Les algorithmes 3 et 4 ont pour idée de demander par un mécanisme d'inondation aux centres de calcul d'évaluer en fonction des caractéristiques de la tâche plusieurs métriques. En fonction des réponses (Algorithme 4) le gestionnaire ayant soumis la tâche va décider à quel site elle sera envoyée pour exécution (Algorithme 3). L'exécution sera alors locale, sans contrôle du gestionnaire de soumission. Ce sera alors le site local qui décidera de la machine exacte pour l'exécution par exemple. On remarquera que ce système est utilisé aussi lorsqu'il s'agit d'un cloud hybride et que des ressources sont disponibles localement. Les métriques envoyées en réponse peuvent contenir des informations de performance, énergétiques, mais aussi de prix d'exécution.

Algorithm 3 Algorithme coopératif de soumission de tâche

1: Proposer la tâche aux voisins
2: Attendre un temps maximum ou d'avoir suffisamment de réponses
3: Trier les réponses reçues en fonction des métriques
4: Soumettre la tâche au meilleur site

6.1.1.2 Re-placement

Comme vu précédemment (Section 3.3.1) plusieurs raisons peuvent provoquer une remise en cause d'un placement pour améliorer les métriques d'un centre de calcul. En

Algorithm 4 Algorithme de réponse

```
1: if Première réception de proposition then
2:     Proposer la tâche aux voisins
3:     Évaluer les machines correspondant aux besoins
4:     Utiliser une heuristique de placement
5:     Répondre en envoyant les métriques
6: end if
```

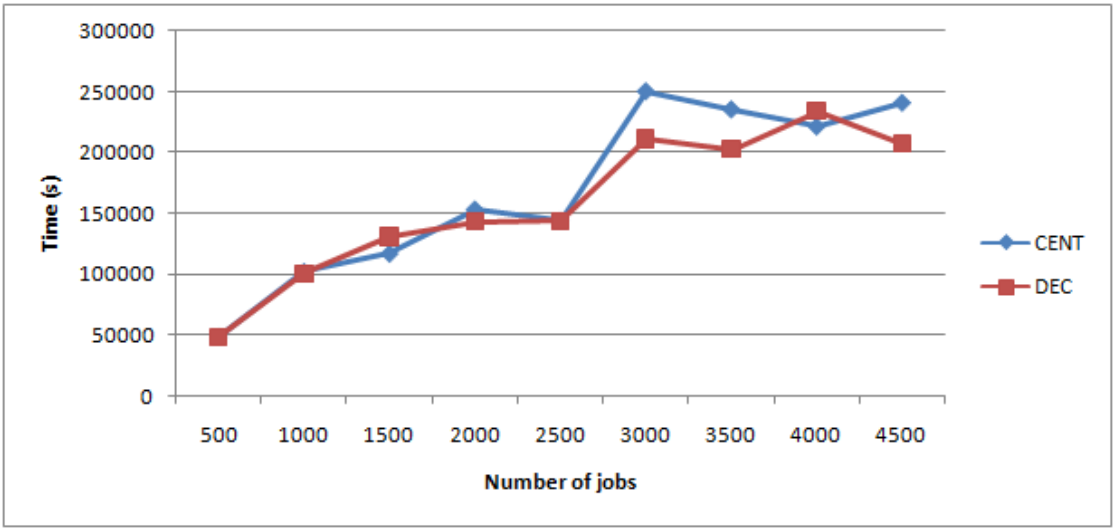


FIGURE 6.2 – Comparaison des algorithmes centralisé et décentralisé coopératif pour le temps d'exécution des tâches

plus de ces raisons, dans le cas d'un système coopératif il peut être intéressant de redistribuer des tâches localement en attente sur d'autre sites. Une tâche peut être mise en attente lorsqu'une tâche locale prioritaire arrive et qu'il n'y a plus de place libre par exemple.

6.1.2 Comparaison entre coopératif et centralisé

Comme décrit en Section 4.2 nous avons étendu les simulateurs CloudSim et MagateSim afin de pouvoir évaluer ce types d'algorithmes coopératifs.

Les figures 6.3 et 6.2 montrent respectivement l'énergie consommée et le temps total d'exécution des tâches pour comparer l'approche coopérative avec une approche centralisée. Pour effectuer cette comparaison, l'algorithme centralisé utilise le même algorithme que celui utilisé pour un site dans l'algorithme coopératif.

Les tâches et les serveurs utilisés ici sont respectivement les tâches de Grid'5000[101] et l'architecture de Grid'5000 (26 sites, 3194 serveurs) obtenues en 2013.

FIGURE 6.3 – Comparaison de l'énergie totale consommée par les tâches gérées par les algorithmes centralisé et décentralisé coopératif

Ces figures montrent un comportement similaire pour la version centralisée et la version coopérative. En effet, vu le grand nombre de serveurs par site et le faible nombre de sites, l'impact de n'utiliser que des vues locales est limité. A grande échelle il est donc raisonnable d'utiliser des algorithmes décentralisés, l'impact étant faible, malgré l'absence de vision globale. En échange, les avantages des systèmes décentralisés permettent une plus grande souplesse et stabilité.

Ce type de système est plus pertinent pour les centres de calcul de taille actuelle que pour l'utilisation pour des centres de calcul très nombreux mais plus petit. En effet, l'algorithme d'inondation qui sert de base à cette approche ne serait plus adapté. De nombreux travaux[102] dans la communauté Pair à Pair seront alors utilisables tels que les algorithmes de bavardage (*Gossip*) ou les algorithmes de marche aléatoire.

Mais au delà de la simple structure de coopération entre les serveurs de coordination, un autre problème se posera. Les bons résultats montrés dans les figures précédentes viennent du fait de la taille des centres de calcul. De par leur taille, les situations de pénuries locales de ressources sont rares. Cette problématique sera essentielle à étudier dans le cas d'un grand nombre de petits centres de calcul en plus de la problématique de la couche de coopération.

6.2 Décisions locales à impact local

Une fois la décision prise du site à utiliser, les algorithmes présentés lors de la section précédente sont utilisables. Mais localement il peut aussi y avoir un très grand nombre de serveurs dans un centre de calcul unique. Il est alors pertinent d'avoir sur chaque serveur un centre de décision capable d'optimiser l'exécution des tâches qui lui sont confiées.

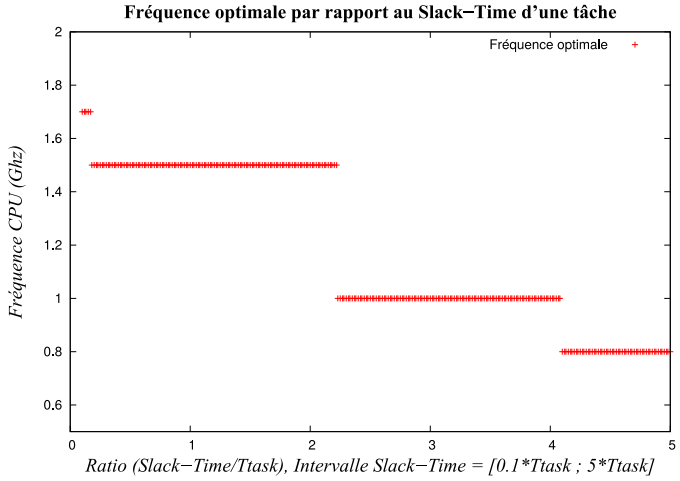


FIGURE 6.4 – Graphique des fréquences optimales en énergie obtenues en utilisant les valeurs de puissances et de fréquences du site Grid'5000 de Reims en fonction du *Slack-Time*. Les valeurs de *Slack-Time* étudiées sont comprises dans l'intervalle $[(0.1 * T_{task}); (5 * T_{task})]$. T_{task} est le temps d'exécution fixé pour une tâche et l'unité de l'axe X est le ratio $\frac{Slack-Time}{T_{task}}$.

Le nombre de métriques à ce niveau est fortement réduit. En effet la majorité des métriques présentées en section 2.3.1 sont applicables au niveau d'un centre de calcul entier. Au niveau local, donc d'un seul serveur, les deux métriques les plus pertinentes sont la performance et l'énergie (ou la puissance). Vu que les décisions sont uniquement locale, il y a une très forte corrélation d'effet entre puissance et température.

L'échelon local ne doit pas être sous-estimé. *Energy Logic* a par exemple montré que la réduction de consommation de 1 Watt au niveau du processeur avait pour effet de réduire la consommation globale de 2.84W[36] du fait des effets de cascades : extraction de la chaleur, pertes lors du transport et de la conversion de l'électricité.

En fonction des informations disponibles, plusieurs granularités sont accessibles.

6.2.1 Gestion du slack

Si on a des informations extérieures sur le temps qu'il est possible d'allouer à une tâche, il peut être intéressant de la ralentir. Par exemple lorsque deux tâches s'exécutent en parallèle sur deux serveurs, que leur temps d'exécution est connu et qu'une est plus longue que l'autre. Cela a un impact sur les différentes métriques, en effet la métrique du temps de réponse change et n'intervient plus comme une métrique mais comme une contrainte.

Par exemple, la Figure 6.4 montre la fréquence optimale en fonction du temps disponible pour économiser l'énergie totale sur un serveur du site de Reims de Grid'5000. Utiliser la fréquence proposée permet de réduire la consommation d'énergie sans impacter la performance globale car il s'agit ici d'utiliser le *Slack-Time* qui n'est pas sur le

TABLE 6.1 – Catégories de phases liées à leur consommation de ressources

Catégorie	Description
Compute intensive	Phases qui demandent beaucoup de calcul ; Leurs performances sont souvent limitées par la vitesse du processeur.
Memory intensive	Phases qui font un grand nombre d'accès en lecture ou écriture à la mémoire.
Mixed	Se réfère aux phases qui partagent les caractéristiques des applications de type *memory intensive* et *compute intensive*.
Communication intensive	Phases qui ont des grands besoins en envoi ou réception réseau et qui sont donc limitées par le matériel réseau ou par des programmes distants.
IO intensive	Phases qui lisent ou écrivent de grandes masses de données localement. Les performances de ces phases dépendent de la vitesse du périphérique idoine.

chemin critique de l'application.

Cette méthode est intermédiaire. Elle nécessite de l'information venant du système global de façon à pouvoir savoir quelle est la borne de ralentissement acceptable. Elle est aussi tributaire de la qualité des informations reçues.

6.2.2 Optimisation à gros grain sans connaissance

Dans le cas où la tâche ne fournit pas de connaissance à priori il devient nécessaire d'utiliser des informations de supervision locale pour en déduire le type de leviers utilisable. Du fait de la latence des choix il est intéressant d'investir dans des décisions ayant un impact sur une relativement longue durée. Les applications ont souvent des parties régulières que l'on appelle phases (voir Section 2.2).

En fonction des ressources utilisées, il sera possible de choisir le levier le plus pertinent. Ainsi une phase d'application n'utilisant que très peu le réseau permettra de baisser le débit maximum de l'interface réseau et donc d'économiser de l'énergie sans impact sur les autres métriques (telle que la performance).

Il est ainsi possible de classifier les phases d'applications en fonction de ces profils (Table 6.1) de consommations de ressources décrits en Section 3.1.1.

Ainsi en fonction du type de ressource consommée, il sera possible d'utiliser un levier particulier. Par exemple, une phase de type *IO-Intensive* sera rarement ralentie par une baisse de la fréquence du processeur. Par contre, une phase de type *Compute-Intensive* dépendra fortement de la fréquence du processeur.

En fonction du type de phase, il est donc possible d'activer des leviers impactant peu la performance mais fortement l'énergie. La table 6.2 rassemble les leviers utilisables en fonction du type de phase.

TABLE 6.2 – Labels des phases et schéma associés de réduction de la puissance

Label des phases	Décision possible de reconfiguration
Compute-intensive	éteindre les bancs de mémoire inutilisés ; mettre les disques en veille ; augmenter la fréquence du processeur ; baisser le débit des interfaces réseaux.
Memory-intensive	réduire la fréquence du processeur ; mettre les disques en mode veille légère ; allumer les bancs mémoires.
Mixed	allumer les bancs mémoire ; augmenter la fréquence du processeur ; mettre les disques en veille ; baisser le débit des interfaces réseaux.
communication intensive	éteindre les bancs mémoire inutilisés ; réduire la fréquence du processeur ; allumer les disques.
IO-intensive	allumer les bancs mémoire ; réduire la fréquence du processeur ; allumer les disques.

Un exemple de gains atteignable avec ce type d'approche est montré en Figure 6.5 pour un cas d'utilisation sur 25 serveurs (100 coeurs) sur Grid'5000. On peut y remarquer que la politique on-demand n'est pas efficace sur deux applications réelles de tests : MDS[12] (Molecular dynamics simulations) et WRF-ARW[92] (Weather Research and Forecasting). En effet, pour ces deux applications, la charge processeur reste constamment à 100%, les comportements on-demand et performance sont donc identiques. Ce comportement est classique[65] pour les applications de type calcul à haute performance. Par contre, entre ces méthodes classiques (performance et on-demand) et notre proposition MREEF (Multi-Resource Energy Efficient Framework[J10]) qui consiste à appliquer les leviers montrés en Table 6.2 pour chaque phase, les gains en énergie sont sensibles (environ 20%) pour une faible perte en temps d'exécution (environ 3%). L'impact sur le temps d'exécution vient du fait qu'en tâche de fond MREEF fait de la détection de phase puis prend la décision de changer les paramètres du système de manière réactive, i.e. sans prédiction.

La difficulté de cette méthode vient de la caractérisation des phases par goulet d'étranglement du type de ressource utilisé. Nous avons vu en section 3.1.1 qu'il est facile de détecter une phase. Par contre il est difficile d'identifier qu'il s'agit d'une phase où la ressource limitante est la mémoire.

De plus même si il est possible de faire de la détection de phase en ligne cela rajoute une latence pour changer la décision prise. Ainsi dans ces deux exemples d'applications, un certain nombre de phases de communication ou d'utilisation disque sont trop rapides (de quelques milli-secondes à quelques secondes) pour être détectées.

Malgré ces inconvénients cette technique a l'avantage de pouvoir être facilement utilisée dans un cadre où les applications et leur comportement sont connus seulement à gros grains et relativement stables. Ainsi cette méthodologie est particulièrement adaptée pour les centres de calcul dédiés au calcul haute performance.

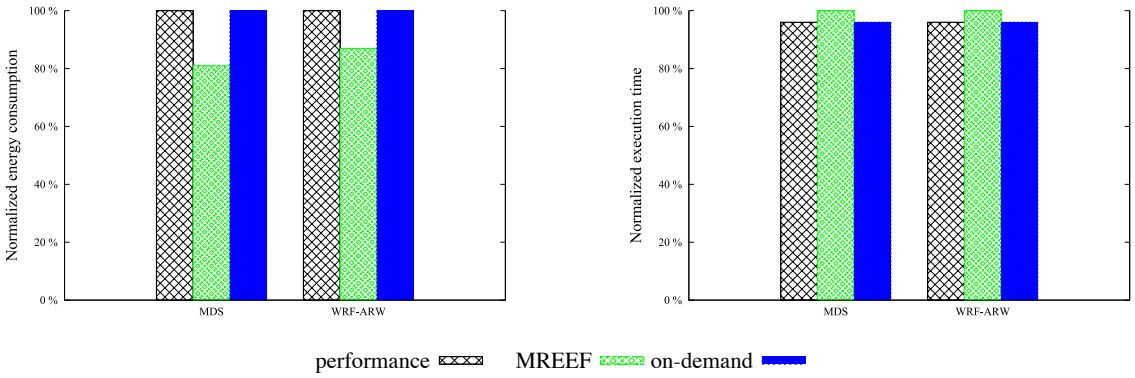


FIGURE 6.5 – Énergie et performance comparées pour les gouverneurs classiques (ondemand et performance) et pour une approche par règles prédéfinies décrites en Table 6.2 (MREEF, *Multi-Resource Energy Efficient Framework*).

6.2.3 Optimisation au niveau noyau

Pour résoudre les deux problématiques du temps de réaction et de la connaissance fine du comportement des applications, il est nécessaire de revenir à la couche système sous-jacente à l'exécution d'une phase ou d'une application.

Au niveau du système d'exploitation, le gouverneur prend plusieurs dizaines de fois par seconde la décision de la fréquence en fonction de la charge. Historiquement, dans le noyau Linux cette valeur est fixée à 10ms. Cette décision est prise pour le pas de temps suivant en fonction de l'état pendant le pas de temps précédent. L'hypothèse est que d'un pas de temps au suivant, le système sera presque toujours le même et qu'ainsi la décision sera valable. C'est sur ce principe que sont construits les gouverneurs *On-demand* et *Conservative* décrits en Section 3.2.2.

Une méthode consiste donc à faire la détection de phase décrite à la section précédente à cette granularité afin de profiter de la vitesse de réaction, mais aussi de la simplicité.

En effet, pendant un pas de temps de quelques milisecondes, il est possible de simplifier le comportement d'une application et de séparer le temps qu'elle passe à faire de la communication, du calcul, des entrées sorties.

Le schéma de la Figure 6.6 montre par exemple une modélisation d'un pas de temps d'une application ne faisant que du calcul et de la communication. Par soucis de clarté, dans la suite de cette section nous nous concentreront uniquement sur ces deux aspects, mais les résultats sont généralisables à d'autres ressources sujettes à contention (accès mémoire, disques).

En utilisant un tel modèle il est possible en fonction du ratio entre communications et calculs de choisir la fréquence optimale pour le processeur, d'un point de vue énergie totale.

La figure 6.7 montre quatre gouverneurs pour les NPB[28] (Nas Parallel Benchmak). Les gouverneurs utilisés sont *performance*, *powersave* et *ondemand* qui ont été décrits

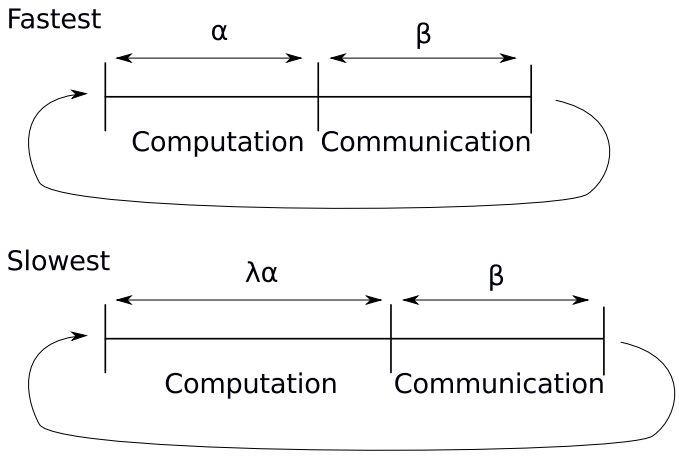


FIGURE 6.6 – Consommation de ressource classique d'une application HPC. Le temps va être $\alpha + \beta$ secondes en utilisant le gouverneur *performance*, et $\alpha\lambda + \beta$ secondes en utilisant *powersave*, avec $\lambda > 1$. λ est le ratio entre les fréquences maximale et minimale du processeur.

en profondeur en section 3.2.2, ainsi que *NetSched* qui utilise la fréquence optimale en fonction du ratio communications/calculs. Ces expériences ont été réalisées sur 4 serveurs du site Toulousain de Grid'5000 avec un pas de temps de $1/10s$.

Il est d'abord intéressant de remarquer que la fréquence a un impact fort sur le temps d'exécution et qu'à fréquence minimale, on peut aller jusqu'à 2.6 fois moins vite et que ce chiffre correspond au ratio entre les fréquences maximale et minimale du processeur utilisé.

Il est particulièrement intéressant de voir que d'un point de vue énergie, *NetSched* permet, suivant les cas, de réduire jusqu'à 25% la consommation énergétique. Ce gain est atteint sans ralentissement (hors pour le programme IS) et même avec une accélération (programmes FT, SP, LU). L'accélération est obtenue grâce à une meilleure utilisation du budget thermique du processeur. En effet, en le ralentissant lorsque c'est possible, il chauffe moins, et il peut donc aller plus vite lorsqu'il y a de forts besoins de calculs. La gestion de l'enveloppe thermique est automatique sur les processeurs de génération récente.

En effet, les phases de communications peuvent parfois être brèves, mais à l'échelle du gouverneur elles sont longues. Les décisions sont alors prises de manière très efficace car adaptées à la tâche en cours.

Un plus grand nombre d'exemples peut être trouvé dans [C10].

6.3 Conclusion

Les centres de calcul vont être de plus en plus distribués. Pour pouvoir continuer à tirer partie de leur apport principal qui est la mutualisation des ressources, la coopé-

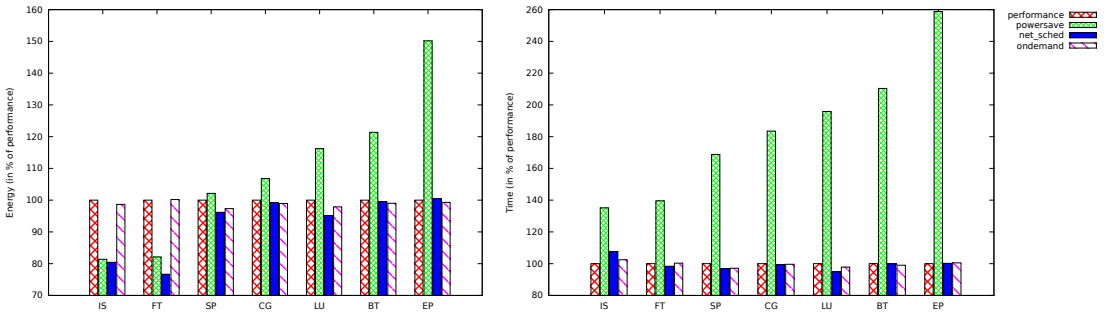


FIGURE 6.7 – Energie moyenne (gauche) et temps d'exécution (droite) des politiques de gestion de la fréquence processeur.

ration entre centres sera nécessaire. L'expérience des grilles de calcul montre que cette coopération est difficile à mettre en place d'un point de vue responsabilité. Les clouds permettent de répondre à cette problématique en séparant fortement les différents niveaux, en formalisant les échanges de ressources par des échanges commerciaux. Grâce à la grande échelle en terme de ressources, on a montré qu'il est possible d'avoir des résultats proches entre une gestion centralisée et une gestion coopérative.

Une piste à explorer consiste à regarder le cas des micro-centres de calcul. En effet dans l'hypothèse des centres de calcul proportionnels, il est probable d'avoir un grand nombre de *data center in the box*. Dans ce cas là, les phénomènes d'échelle seront moins présents et donc une gestion coopérative plus fine sera nécessaire.

Enfin, du fait du grand nombre de centres de calcul, il sera aussi nécessaire de faire évoluer les algorithmes coopératifs, car à présent ils reposent sur un principe d'inondation, et donc limité à un petit nombre de centres.

A l'autre bout de l'échelle, un grand travail reste à faire au niveau local. En effet, la modélisation fine du comportement d'une application en fonction de l'état du serveur sur laquelle elle s'exécute est encore trop légère. Notre première approche qui consiste à modéliser une application comme une succession de calculs et de communication permet de lever ce voile. En utilisant cette modélisation et des changements de fréquence adaptés, nous avons réussi à économiser de l'énergie sans sacrifier la performance. Un grand travail reste à faire pour inclure les autres sous-systèmes (mémoire, disque,...) dans cette modélisation. Un chantier encore plus grand consiste à étudier le comportement de plusieurs applications simultanées. Plusieurs travaux[91, 48, 69] montrent que l'isolation entre machines virtuelles n'est pas parfaite, plus particulièrement au niveau performance.

Le dernier élément concerne le lien entre les deux niveaux. La proposition actuelle d'utiliser seulement deux niveaux est raisonnable de manière limitée du fait de la petite échelle du nombre d'acteurs. En effet, la majorité des travaux considère uniquement une fédération de centres de calcul tous gérés par la même entité ou l'utilisation d'un faible nombre de clouds commerciaux. Dans tous les cas, leur nombre est limité. Lorsque le

nombre de centre de calcul ainsi que leur hétérogénéité seront décuplés, les techniques actuelles (telles que les Super-Peers) montreront leurs limites. Une porte de sortie consistera à utiliser des méthodes plus orientées multi-agent par exemple.

Chapitre 7

Conclusion et projet de recherche

> Je ne cherche pas à connaître les réponses, je cherche à comprendre les questions.
>
> Confucius

La question initiale était : *Comment gérer efficacement un centre de calcul ?* Ce mémoire présente l'ensemble des éléments nécessaires pour répondre à cette question, de la formalisation de ses termes jusqu'à la proposition d'algorithmes permettant cette gestion.

7.1 Conclusion

Le sens de la question

Pour y répondre il est nécessaire de définir précisément ce qu'est un centre de calcul. Dans ces travaux nous avons défini les centres de calcul mais aussi chacun de leurs éléments avec différent niveaux de précision. Mais au delà de la précision dans l'absolu, nous avons montré qui est nécessaire d'avoir une précision raisonnable et équilibrée pour l'ensemble de la modélisation, et donc des modélisations de complexités et de précisions équivalentes pour chacun des éléments.

Un grand nombre d'approches sans modélisation précise font l'impasse sur la problématique de la mesure : qualité du matériel de mesure, mais aussi modélisation des pertes dans les alimentations. Nous avons montré que dans un grand nombre d'études ces éléments peuvent être éludés, mais uniquement parce que nous en donnons les clés permettant de le faire en connaissance de cause, et d'en maîtriser l'impact sur les résultats expérimentaux.

Le domaine des métriques est lui aussi complexe. Il est possible de faire dire des choses fausses aux métriques agrégées. Il n'est pas possible de faire l'économie d'exprimer les métriques simples (Energie, Performance, Robustesse,...). Nous avons aussi montré que dans certains cas les métriques composées (PUE) donnent des résultats simplement faux.

Enfin nous avons montré dans quels cas et pour quels usages elles sont pertinentes et valides.

Le monde réel

Nous avons ensuite abordé le problème de l'implémentation effective des leviers et des capteurs. Nous avons proposé des capteurs évolués au niveau des applications permettant de déterminer les phases ainsi que leur caractéristiques. Les leviers étudiés permettent d'agir sur le monde en connaissant leur impact, mais aussi leur latence d'action ou tout autre sur-coût. Au delà de l'utilisation de ces capteurs et leviers, la grande question est l'instant de décision. Nous avons étudié l'impact des constantes de temps (dynamisme des charges, temps d'allumage de serveur,...) sur le choix du mode de déclenchement : événementiel (nouvelle machine virtuelle) ou régulier.

Les outils

Pour évaluer il existe plusieurs outils, allant de la modélisation directe à l'expérimentation en passant par la simulation. Pour cette dernière catégorie, les simulateurs existants ont dû être augmentés pour être capables de prendre en compte les modèles fins de consommation électrique, mais aussi de l'effet des leviers tels que la gestion de fréquence des processeurs.

La gestion efficace

Un certain nombre d'heuristiques ont été ensuite étudiées. Plus particulièrement, le rôle des tris de serveurs et de tâches qui ont un impact sur l'ensemble des heuristiques. De même, la formulation exacte du multi-objectif a un impact fort, pas seulement sur le résultat mais aussi pour l'utilisateur final. Nous avons exploré l'impact du temps, autant d'un point de vue inertie thermique que d'un point de vue latence d'utilisation des leviers.

Enfin, nous avons montré qu'il est possible d'utiliser des algorithmes coopératifs pour gérer des fédérations de centres de calcul. Il est aussi possible d'avoir de forts gains énergétiques sans perte de performance en ne prenant des décisions qu'au niveau des serveurs eux même. Ces décisions étant locales, elles sont utilisables à très grande échelle.

L'humain

Le point de vue développé ici est celui de l'opérateur. Pourtant pour avoir une vision plus complète il est aussi nécessaire de tenir compte de l'effet de la gestion fine de l'énergie sur lui. En effet, ne faire que réduire la consommation énergétique des centres de calcul n'est pas suffisant. Le paradox de Jevons[52, 44] (effet rebond) montre que lorsqu'on optimise la consommation énergétique, souvent l'effet est de produire plus de travail pour la même consommation. C'est pourquoi le rôle du législateur est particulièrement important même si il n'a pas été abordé. Son rôle peut être autant de fixer une limite sur la consommation totale ou sur l'efficacité énergétique, que de simplement forcer à informer les utilisateurs finaux de leur impact environnemental.

7.2 Projet de recherche

Ce document retrace les principales recherches auxquelles j'ai participé depuis mon arrivée à Toulouse. Ces recherches, autant simulation qu'en expérimentation se structurent autour de la boucle de gestion autonome d'un centre de calcul : capteurs, effecteurs, système de décision, mais aussi modélisation du système. A court terme il reste un grand nombre d'éléments manquant pour généraliser et approfondir les résultats déjà obtenus.

7.2.1 Projet de recherche à court terme

A court termes, quatre directions seront abordées dans les prochaines thèses :

Niveau système Comme décrit en Section 2.2, une des hypothèses simplificatrices le plus souvent utilisée au niveau système consiste à supposer que les applications n'ont pas d'interactions entre elles. Une piste de recherche consiste alors à modéliser plus finement cette interaction autour de la notion de goulet d'étranglement. Obtenir ce type de modèle nous permettra d'avoir autant un modèle plus fin de ralentissement du fait des interactions entre applications, mais surtout d'avoir un modèle plus fin de prédiction de comportement, de consommation de ressources et donc d'énergie. Enfin ce modèle permettra de raffiner le traducteur de profils en croisant le modèle fin d'une application, celui du serveur où ce profil a été récupéré ainsi que celui du serveur de destination. Enfin, il sera possible de généraliser ce traducteur, en accord avec les modèles de leviers matériels (DVFS par exemple) pour prendre des décisions locales plus fines comme celles abordées en Section 6.2.

Niveau apprentissage De manière intermédiaire entre le niveau système et le niveau apprentissage un élément important que nous allons devoir gérer consiste en la création automatique de micro-programmes de couverture maximale (Section 3.1.3). Pour cela, il sera nécessaire de pouvoir utiliser des bibliothèques de découverte automatique du matériel. En utilisant ces informations, la question sera alors, en fonction du besoin de couverture, d'être capable de générer de manière automatique un ensemble minimal (en temps d'exécution) de programmes de couverture maximale. Pour atteindre ce résultats une méthode basée sur des algorithmes génétiques utilisant des petits corpus de codes seront évalués.

Niveau métriques L'étude des métriques, de leur limites et la classification de métriques en fonction de leur cadre d'utilisation (Section 2.3.1) reste un domaine complexe. Dans le cadre du *Cluster on renewable-energy based data-center*[1] dans lequel je participe à la suite de CoolEmAll, nous avons étudié[RT12] la capacité des métriques classiques de centres de calcul à évaluer la prise en compte d'énergie renouvelables. Pour cela 36 métriques ont été analysées et finalement 25 nouvelles métriques ont été proposées et sont en cours d'évaluation. Ces chiffres montrent

1. Regroupe les membres des projets européens DC4Cities, GENiC, CoolEmAll, RenewIT, GEYSER, GreenDataNet, doLFin et All4green

qu'il est difficile d'obtenir un ensemble restreint d'indicateurs clés permettant de comparer des centres de calculs (au niveau infrastructure, intergiciel, système). Nous prévoyons de continuer dans cette voie afin de fournir un ensemble réduit et pertinent d'indicateurs tenant compte de l'environnement des centres de calculs.

Niveau Optimisation Dans ce mémoire nous avons étudiés certaines des caractéristiques des heuristiques permettant une utilisation efficace d'un centre de calcul. Historiquement ces travaux prennent leur source dans une vision assez stable de la charge de travail. La prochaine étape consistera à étudier plus finement la temporalité et son impact. Comment évolue le système en fonction du temps entre deux décisions et en fonction des événements forçant à prendre une décision (sujet abordé en Section 3.3.1). L'autre élément étudié sera l'impact de la qualité des données de supervision sur le système de décision. Cette qualité peut être évaluée en précision, synchronisation et âge des mesures (Section 3.1). D'un point de vue (meta)-heuristique, les pistes d'amélioration ont rapidement été évoquées en Section 5. Les problématiques à résoudre concernent autant le placement que les migrations ou la politique de gestion de l'allumage et de l'extinction des serveurs. Un noeud de recherche particulièrement intéressant concernera la coordination des fédérations de centres de calcul (Section 6.1).

Il existe aussi des pistes d'ingénierie ouvertes comme l'intégration de l'ensemble des modèles (de matériels et d'applications) et scénarios dans un seul simulateur, avec la possibilité de tester le même système en simulation mais aussi sur plateforme réelle (Section 4). Les travaux actuels (plus particulièrement ceux d'Inès) s'orientent vers la co-simulation ainsi que la simulation hybride. Un exemple de co-simulation consiste à prendre un simulateur de centres de calcul tels que DCWorms, et à le coupler avec un simulateur de panneaux solaires et un simulateur de batteries. Le simulateur RenewSim est en fait un co-simulateur de centre de calculs interfacés avec du stockage et de la production d'électricité solaire. La simulation hybride consiste à utiliser des capteurs simulés mais aussi des capteurs réels dans une infrastructure réelle. On peut alors utiliser des capteurs réels sur un panneau solaire couplés avec une simulation d'un centre de calcul.

Deux grandes pistes complémentaires de recherche de plus long terme sont particulièrement intéressantes : l'intégration des centres de calcul avec leur environnement ; la vision multi-agent de leur gestion.

7.2.2 Les centres de calcul et leur environnement

Jusqu'à présent, la majorité des études portant sur l'optimisation des centres de calcul les considèrent comme des objets éthérés, hors sol. L'hypothèse est alors que l'impact des interactions avec l'extérieur est faible par rapport au comportement interne.

Pourtant trois phénomènes rendent caduque cette hypothèse :

— La multiplication des petits centres de calcul qui sont donc plus densément intégrés à leur environnement (*data center in the box*, radiateur liés au cloud[64]) ;

— L'augmentation des liens avec l'environnement (*free cooling*, réutilisation de la chaleur) ;
— L'utilisation de plus en plus importante des énergies renouvelables, intermittentes par définition.

Concernant l'augmentation de l'importance des énergies renouvelables intermittentes, l'exemple de l'éolien est représentatif : Même si en 2008 l'énergie éolienne ne représentait que 1% de l'électricité produite aux US[96] et 4.5% fin 2013[106], le *Department of Energy* y a fixé un objectif de 20% pour 2030[96].

Plusieurs directions sont nécessaires pour étudier ces phénomènes :
— Une modélisation précise des interactions ;
— Des métriques adaptées tenant compte des services rendus par les centres de calcul ;
— L'intégration dans la notion de qualité de service de la plasticité des tâches ;
— L'adaptation des modèles économiques.

En effet, un centre de calcul n'a pour l'instant qu'une capacité d'adaptation légère, son but étant d'exécuter au mieux les tâches qui lui sont soumises. En interaction avec son environnement, il va devoir gérer ses tâches, peut être les reconfigurer ou les décaler dans le temps, afin de satisfaire ces tâches mais aussi de rendre service à son environnement.

Dans le cas d'énergie renouvelable par exemple, le centre de calcul va gérer les tâches de façon à ce que sa consommation électrique suive au mieux la production d'énergie. Il faut alors avoir assez d'informations sur les tâches pour les adapter, ou même simplement les mettre en pause.

7.2.3 Gestion par multi-agents de centres de calcul

Le système actuel fonctionne car il y a un faible nombre de très gros opérateurs, quelques opérateurs moyens est des infrastructures internes dans les entreprises. Dans ce cadre là, les utilisateurs décident l'utilisation de ressources leur appartenant ou distantes, puis les gros opérateurs décident dans quel centre de calcul exécuter les machines virtuelles. Enfin, dans le centre de calcul, un ordonnanceur global décide de l'ensemble des opérations (gestion des tâches, machines,...).

Ce schéma global, possible pour un nombre restreint d'acteurs ne fonctionne plus lorsque ce nombre change d'échelle. Dans un futur proche, on s'oriente vers un grand nombre de centres de calcul mais aussi d'opérateurs. Le constat[37] de Gartner [2] va dans le sens de l'explosion du nombre de centres de calcul de tailles diverses, plus particulièrement des petits : "The recent trend to centralize applications to reduce costs and increase security is incompatible with the IoT. Organizations will be forced to aggregate data in multiple distributed mini data centers where initial processing can occur. Relevant data will then be forwarded to a central site for additional processing". De même, IDC [3] prévoit[86] que d'ici 2018 le nombre de centre de calcul et surtout d'opérateurs va largement augmenter. Enfin, pour des raisons de gestion de données privées, on s'attend

2. `http://www.gartner.com/`
3. `http://www.idc.com/`

aussi à voir se multiplier les Cloud personnels[23].

Deux éléments seront nécessaires pour pouvoir gérer une telle multiplicité de systèmes : une capacité de coopération entre centres de calcul ; différent niveaux de gestion.

7.2.3.1 Coopération

Des standards[80] tels que *Simple Cloud API*, OCCI (*Open Cloud Computing Interface*) ou plus récemment CIMI (*Cloud Infrastructure Management Interface*) permettent de gérer la partie technique concernant l'interopérabilité. Pourtant même avec ces bibliothèques haut niveau, nous sommes encore bien loin de l'idée originelle d'avoir un accès à des ressources informatique ayant la même simplicité qu'un accès à l'eau ou à l'électricité.

Cette simplicité vient de structures intermédiaires qui coordonnent un grand nombre de sites de production, donnant une apparence d'unicité de l'interlocuteur inexistante dans le cas des centres de calcul. Il serait possible que des entreprises prennent en charge ce rôle, fournissant un centre de calcul virtualisé reposant sur un grand nombre de prestataires possédant réellement les ressources.

Une autre méthode consiste à établir des liens entre les centres de calcul, leur permettant de coopérer en mettant en commun certaines ressources lorsque le besoin s'en fait sentir. Ils pourraient ainsi échanger des machines virtuelles par exemple. La majorité des propositions dans ce sens reposent sur des tiers de confiance. Une méthode intéressante serait d'étudier le cas où les opérateurs tissent un réseau de confiance, de gré à gré.

7.2.3.2 Multi-agent

Mais au niveau de chaque centre de calcul, la situation est bien plus complexe. Dans un tel système, des contraintes différentes viendraient des opérateurs voisins, mais aussi des contraintes environnementales (telles que l'approvisionnement en électricité).

Au niveau de chaque élément (serveurs, armoires de serveurs,...) l'équation serait aussi complexe. Chacun de ces éléments devrait gérer à la fois les ordres qui lui sont envoyés, mais aussi les sous-systèmes en dessous de lui.

Par exemple, un serveur doit à la fois gérer les machines virtuelles qu'il doit exécuter, mais aussi la fréquence de son processeur et le débit de son interface réseau.

Enfin, nous avons vu que les interactions peuvent être complexes, telle que celle concernant la production et l'extraction de la chaleur.

Avec un tel nombre de contraintes, il devient impossible d'optimiser toutes les métriques dans un centre de calcul.

Une solution serait d'utiliser un système d'agents où chaque élément (machine virtuelle, serveur, armoire,...) serait capable de prendre des décisions simples. Ces agents pourraient alors prendre des décisions sur des informations partielles, tenant compte des priorités locales. Un système de production électrique prévoyant une baisse de son apport pourrait envoyer une *intention* aux agents avec lesquels il coopère afin de mieux répartir la production de chaleur.

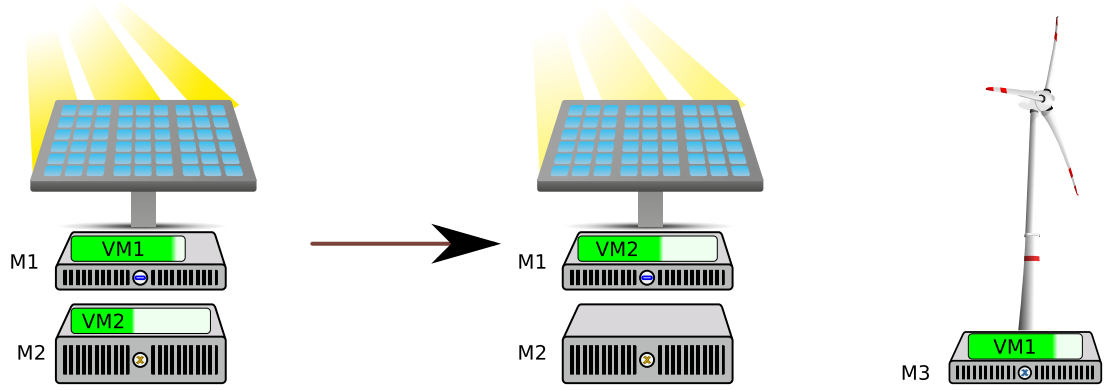


FIGURE 7.1 – Scénario de coopération entre agents pour la gestion d'un centre de calcul

Un autre exemple est montré en Figure 7.1. Dans cet exemple, plusieurs agents coopèrent. Un agent gère le panneau solaire, trois autres chacune des machines, et enfin deux gèrent les machines virtuelles. La coopération entre ces différents agents permet d'adapter l'exécution des VMs en fonction de leurs contraintes de QoS et des ressources disponibles sans nécessiter d'avoir un algorithme centralisé. Grâce à cette méthode il devient possible de gérer de manière coopérative une fédération à très grande échelle de systèmes tout en tenant compte d'un grand nombre de contraintes (ici de QoS, énergétiques et thermiques).

Une fois ce modèle de coopération à base d'agents défini, il est extensible à d'autres systèmes à bases de ressources où les éléments consommant les ressources sont avec des contraintes de qualité de services différentes. On peut simplement établir un parallèle avec les bâtiments ou les villes intelligentes.

Publications

Journaux

[J1] Damien Borgetto, Henri Casanova, Georges Da Costa et Jean-Marc Pierson. "Energy-Aware Service Allocation". In : *Future Generation Computer Systems* 28.5 (2012). Special section on Energy Efficiency in Large-Scale Distributed Systems, p. 769–779. URL : http://dx.doi.org/10.1016/j.future.2011.04.018.

[J2] Leandro Cupertino, Georges Da Costa, Ariel Oleksiak, Wojciech Pia, Jean-Marc Pierson, Jaume Salom, Laura Sisó, Patricia Stolf, Hongyang Sun, Thomas Zilio et al. "Energy-efficient, thermal-aware modeling and simulation of data centers : The CoolEmAll approach and evaluation results". In : *Ad Hoc Networks* 25 (2015), p. 535–553. URL : http://dx.doi.org/10.1016/j.adhoc.2014.11.002.

[J3] Mehdi. Diouri, Ghislain Landry Tsafack Chetsa, Olivier Gluck, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf et Georges Da Costa. "Energy efficiency in HPC : with or without knowledge on applications and services". In : *International Journal of High Performance Computing Applications* (2013), p. 232–243. URL : http://dx.doi.org/10.1177/1094342013497990.

[J4] Leandro Fontoura Cupertino, Georges Da Costa et Jean-Marc Pierson. "Towards a generic power estimator". In : *Computer Science - Research and Development, Ena-HPC 2014* 30-2 (2015), p. 145–153. URL : http://dx.doi.org/10.1007/s00450-014-0264-x.

[J5] Leandro Fontoura Cupertino, Georges Da Costa, Amal Sayah et Jean-Marc Pierson. "Valgreen : an Application's Energy Profiler". In : *International Journal of Soft Computing and Software Engineering, SCSE'13 conference* 3.3 (2013). URL : http://dx.doi.org/10.7321/jscse.v3.n3.83.

[J6] Da Costa Georges et al. "Exascale machines require new programming paradigms andruntimes". In : *Supercomputing Frontiers and Innovations* (2015).

[J7] Tom Guerout, Georges Da Costa, Thierry Monteil, Rodrigo Neves Calheiros, Rajkumar Buyya et Mihai Alexandru. "Energy-aware simulation with DVFS". In : *Simulation Modelling Practice and Theory, Energy efficiency*

in Grids and Clouds 39 (2013), p. 76–91. URL : http://dx.doi.org/10.1016/j.simpat.2013.04.007.

[J8] Tom GUEROUT, Samir MEDJIAH, Georges DA COSTA et Thierry MONTEIL. "Quality of Service Modeling for Green Scheduling in Clouds". In : *Sustainable Computing* (2014). URL : http://dx.doi.org/10.1016/j.suscom.2014.08.006.

[J9] Hongyang SUN, Patricia STOLF, Jean-Marc PIERSON et Georges DA COSTA. "Energy-efficient and thermal-aware resource management for heterogeneous datacenters". In : *Sustainable Computing : Informatics and Systems* (2014), p. 1–15. URL : http://dx.doi.org/10.1016/j.suscom.2014.08.005.

[J10] Ghislain Landry TSAFACK CHETSA, Laurent LEFÈVRE, Jean-Marc PIERSON, Patricia STOLF et Georges DA COSTA. "Exploiting performance counters to predict and improve energy performance of HPC systems". In : *Future Generation Computer Systems* (2014), p. 287–298. URL : http://dx.doi.org/10.1016/j.future.2013.07.010.

[J11] Violaine VILLEBONNET, Georges DA COSTA, Laurent LEFÈVRE, Jean-Marc PIERSON et Patricia STOLF. "Big, Medium, Little : Reaching Energy Proportionality with Heterogeneous Computing Scheduler". In : *Parallel Processing Letters* (2015).

[J12] PiaTek WOJCIECH, Oleksiak ARIEL et Da Costa GEORGES. "Energy and thermal models for simulation of workload and resource management in computing systems". In : *Simulation Modelling Practice and Theory* (2015).

Conférences

[C1] Damien BORGETTO, Georges DA COSTA, Jean-Marc PIERSON et Amal SAYAH. "Energy-Aware Resource Allocation (regular paper)". anglais. In : *Energy Efficient Grids, Clouds and Clusters Workshop (co-located with Grid) (E2GC2), Banff, 13/10/2009-15/10/2009*. IEEE, 2009.

[C2] Damien BORGETTO, Michael MAURER, Georges DA COSTA, Ivona BRANDIC et 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*. ACM DL, 2012.

[C3] Nicolas CAPIT, Georges DA COSTA, Yannis GEORGIOU, Guillaume HUARD et Cyrille MARTIN. "A batch scheduler with high level components". In : *IEEE International Symposium on Cluster Computing and the Grid (CCGrid), Cardiff, UK, 09/05/2005-12/05/2005*. IEEE, 2005, p. 776–783.

[C4] Georges Da Costa. "Heterogeneity : The Key to Achieve Power-Proportional Computing (regular paper)". In : *IEEE International Symposium on Cluster Computing and the Grid (CCGrid), Delft, 13/05/13-16/05/13*. IEEE, 2013, p. 656–662. URL : http://doi.ieeecomputersociety.org/10.1109/CCGrid.2013.90.

[C5] Georges Da Costa, Marcos Dias De Assuncao, Jean-Patrick Gelas, Yannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard et Amal Sayah. "Multi-Facet Approach to Reduce Energy Consumption in Clouds and Grids : The GREEN-NET Framework (regular paper)". In : *ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy), Passau, Germany, 13/04/2010-15/04/2010*. ACM, 2010, p. 95–104. URL : http://portal.acm.org/toc.cfm?id=1791314&type=proceeding&coll=portal&dl=ACM.

[C6] Georges Da Costa, Marios D. Dikaiakos et Salvatore Orlando. "Nine months in the life of EGEE : a look from the South". In : *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Istanbul, 24/10/2007-26/10/2007*. IEEE, 2007, p. 281–287.

[C7] Georges Da Costa, Guillaume Dufour et David Sanchez. "Modèles fluides pour l'économie d'énergie dans les grilles par migration : une première approche (regular paper)". In : *Rencontres francophones du Parallélisme (Renpar), Toulouse, 09/09/2009-11/09/2009*. 2009.

[C8] Georges Da Costa, Jean-Patrick Gelas, Yannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard et Kamal Sharma. "The green-net framework : Energy efficiency in large scale distributed systems (regular paper)". In : *High Performance Power Aware Computing Workshop in conjunction with IPDPS 2009 (HPPAC), Rome, 25/05/2009-29/05/2009*. IEEE Computer Society, 2009, p. 1–8.

[C9] Georges Da Costa et Helmut Hlavacs. "Methodology of Measurement for Energy Consumption of Applications (regular paper)". In : *Energy Efficient Grids, Clouds and Clusters Workshop (co-located with Grid) (E2GC2), Brussels, 25/10/2010-29/10/2010*. IEEE, 2010.

[C10] Georges Da Costa et Pierson Jean-Marc. "DVFS governor for HPC : Higher, Faster, Greener (regular paper)". In : *Euromicro International Conference on Parallel, Distributed and network-based Processing, Turku, Finland, 04/03/2015-06/03/2015*. CPS (Conference Publishing Services), 2015.

[C11] Georges Da Costa, Ariel Oleksiak, Wojciech Piątek, Jaume Salom et Laura Siso. "Minimization of Costs and Energy Consumption in a Data Center by a Workload-Based Capacity Management (regular paper)". In : *International Workshop on Energy-Efficient Data Centres, co-located with E-Energy (E2DC), Cambridge, 10/06/2014*. Springer, 2014, p. 102–119. URL : http://link.springer.com/chapter/10.1007/978-3-319-15786-3_7.

[C12] Georges Da Costa, Aurelien Ortiz et Ronan Guivarch. "Add Green in your numerical recipes for a more tasteful world (regular paper)". In : *International Conference on Computer Science and Information Technology (CSIT), Yeravan, 24/07/2011-25/07/2011*. National Academy of Science of Armenia, 2011, p. 240–243.

[C13] Georges Da Costa et Jean-Marc Pierson. "Characterizing applications from power consumption : A case study for HPC benchmarks (short paper)". In : *International Conference on ICT as Key Technology for the Fight against Global Warming (ICT-GLOW), Toulouse, 29/08/2011-02/09/2011*. Springer, 2011.

[C14] Georges Da Costa, Thomas Zilio, Mateusz Jarus et Ariel Oleksiak. "Energy- and Heat-aware HPC Benchmarks (regular paper)". In : *Int'l Workshop on European Actions Towards Eco-Friendly Data Centers (EuroEcoDc), Co-Located with Cloud and Green Computing (CGC), Karlsruhe, 30/09/2013-02/10/2013*. IEEE, 2013.

[C15] Leandro Fontoura Cupertino, Georges Da Costa, Amal Sayah et Jean-Marc Pierson. "Energy Consumption Library (short paper)". anglais. In : *Energy Efficiency in Large Scale Distributed Systems (EE-LSDS), Vienna, Austria, 22/04/2013-24/04/2013*. Sous la dir. de Jean-Marc Pierson, Georges Da Costa et Lars Dittmann. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2013, p. 51–57. url : `http://dx.doi.org/10.1007/978-3-642-40517-4_4`.

[C16] Helmut Hlavacs, Georges Da Costa et Jean-Marc Pierson. "Energy Consumption of Residential and Professional Switches (regular paper)". In : *International Conference on Computational Science and Engineering (CSE), Vancouver, Canada, 29/08/2009-31/08/2009*. IEEE Computer Society, 2009, p. 240–246.

[C17] Hongyang Sun, Patricia Stolf, Jean-Marc Pierson et Georges Da Costa. "Multi-Objective Scheduling for Heterogeneous Server Systems with Machine Placement (regular paper)". In : *IEEE International Symposium on Cluster Computing and the Grid (CCGrid), Chicago, 26/05/2014-29/05/2014*. IEEE Computer Society, 2014. url : `http://oatao.univ-toulouse.fr/12925/`.

[C18] Cheikhou Thiam et Georges Da Costa. "Anti-Load Balancing to Reduce Energy Consumption (student paper)". In : *International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Ajaccio-Corsica-France, 13/01/2011-15/04/2011*. Sous la dir. de P Iványi et B H V Topping. Civil-Comp Proceedings, 2011. url : `http://www.ctresources.info/ccp/paper.html?id=6278`.

[C19] Cheikhou Thiam, Georges Da Costa et Jean-Marc Pierson. "Cooperative Scheduling Anti-load balancing Algorithm for Cloud : CSAAC (short paper)". In : *IEEE International Conference on Cloud Computing Technology and Science, Bristol, UK, 02/12/2013-05/12/2013*. IEEE, 2014, p. 433–438. url : `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6753828`.

[C20] Cheikhou Thiam, Georges Da Costa et Jean-Marc Pierson. "Energy aware clouds scheduling using anti-load balancing algorithm : EACAB (short paper)". In : *International Conference on Smart Grids and Green IT Systems (SMARTGREENS), Barcelona, SPAIN, 03/04/2014-04/04/2014*. INSTICC - Institute for Systems, Technologies of Information, Control et Communication, 2014, p. 0–0.

[C21] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf et Georges Da Costa. "A Runtime Framework for Energy Efficient HPC Systems Without a Priori Knowledge of Applications (regular paper)". In : *IEEE International Conference on Parallel and Distributed Systems, Singapore, 17/12/2012-19/12/2012*. IEEE Computer Society - Conference Publishing Services, 2012, p. 660–667.

[C22] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf et Georges Da Costa. "A User Friendly Phase Detection Methodology for HPC Systems' Analysis (regular paper)". In : *IEEE International Conference on Green Computing and Communications, Beijing (Chine), 20/08/2013-23/08/2013*. IEEE Computer Society, 2013, p. 118–125.

[C23] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf et Georges Da Costa. "Application-Agnostic Framework for Improving the Energy Efficiency of Multiple HPC Subsystems (regular paper)". In : *Parallel, Distributed, and Network-Based Processing (PDP), Turku, Finland, 04/03/2015-06/03/2015*. CPS (Conference Publishing Services), 2014.

[C24] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf et Georges Da Costa. "Beyond CPU Frequency Scaling for a Fine-grained Energy Control of HPC Systems (regular paper)". In : *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), New York, 24/10/2012-26/10/2012*. IEEE Computer Society - Conference Publishing Services, 2012, p. 132–138. URL : `http://doi.ieeecomputersociety.org/10.1109/SBAC-PAD.2012.32`.

[C25] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf et Georges Da Costa. "DNA-Inspired Scheme for Building the Energy Profile of HPC Systems (regular paper)". In : *International Workshop on Energy-Efficient Data Centres, Madrid, 08/05/2012-08/05/2012*. Springer, 2012, p. 141–152. URL : `http://dx.doi.org/10.1007/978-3-642-33645-4_13`.

[C26] Violaine Villebonnet et Georges Da Costa. "Thermal-aware cloud middleware to reduce cooling needs (regular paper)". In : *IEEE International Conference on Collaboration Technologies and Infrastructures (WETICE), Parma, Italy, 23/06/2014-25/06/2014*. CPS (Conference Publishing Services), 2014, p. 115–120.

[C27] Violaine Villebonnet, Georges Da Costa, Laurent Lefèvre, Jean-Marc Pierson et Patricia Stolf. “Towards Generalizing "Big.Little" for Energy Proportional HPC and Cloud Infrastructures (regular paper)”. In : *IEEE International Conference on Sustainable Computing and Communications (SustainCom), Sydney, Australia, 03/12/2014-05/12/2014*. CPS (Conference Publishing Services), 2014.

[C28] Micha Vor Dem Berge, Wolfgang Christmann, Eugen Volk, Stefan Wesner, Ariel Oleksiak, Tomasz Piontek, Georges Da Costa et Jean-Marc Pierson. “CoolEmAll - Models and Tools for Optimization of Data Center Energy-efficiency (regular paper)”. In : *IFIP Conference on SustainIT (Sustainable Internet and ICT for Sustainability), Pisa, 04/10/2012-05/10/2012*. IFIP, 2012.

[C29] Micha Vor Dem Berge, Georges Da Costa, Mateusz Jarus, Ariel Oleksiak, Wojciech PiąTek et Eugen Volk. “Modeling Data Center Building Blocks for Energy-efficiency and Thermal Simulations (regular paper)”. In : *International Workshop on Energy-Efficient Data Centres, co-located with E-Energy (E2DC), Berkeley, 21/05/2013*. Springer, 2013.

[C30] Micha Vor Dem Berge, Georges Da Costa, Andreas Kopecki, Ariel Oleksiak, Jean-Marc Pierson, Tomasz Piontek, Eugen Volk et Stefan Wesner. “Modeling and Simulation of Data Center Energy-Efficiency in CoolEmAll (regular paper)”. In : *International Workshop on Energy-Efficient Data Centres, co-located with E-Energy (E2DC), Madrid, 08/05/2012*. T. 7396. Springer, 2012, p. 25–36.

Chapitres

[Ch1] Pragati Agrawal, Damien Borgetto, Carmela Comito, Georges Da Costa, Jean-Marc Pierson, Payal Prakash, Shrisha Rao, Domenico Talia, Cheikhou Thiam et Paolo Trufio. “Scheduling and Resource Allocation”. In : *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*. Sous la dir. de Jean-Marc Pierson. ISBN: 978-1-118-86463-0. John Wiley et Sons, 2015. Chap. 8, p. 225–262.

[Ch2] Robert Basmadjian, Georges Da Costa, Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Ariel Oleksiak et Jean-Marc Pierson. “Energy-Aware Approaches for HPC Systems”. In : *High-Performance Computing on Complex Environments*. Sous la dir. d’Emmanuel Jeannot et Julius Zilinskas. ISBN: 978-1-118-71205-4. Wiley, 2014. Chap. 18, p. 343–361.

[Ch3] Robert Basmadjian et al. “Green Data Centers”. In : *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*. Sous la dir. de Jean-Marc Pierson. ISBN: 978-1-118-86463-0. John Wiley et Sons, 2015. Chap. 6, p. 159–196.

[Ch4] Micha vor dem Berge, Georges Da Costa, Mateusz Jarus, Ariel Oleksiak, Wojciech Piatek et Eugen Volk. "Modeling data center building blocks for energy-efficiency and thermal simulations". In : *Energy-Efficient Data Centers*. ISBN: 978-3-642-33644-7. Springer, 2014, p. 66–82.

[Ch5] Damien Borgetto, Henri Casanova, Georges Da Costa et Jean-Marc Pierson. "Energy-Efficient Job Placement in Clusters, Grids and Clouds". In : *Energy-Efficient Distributed Computing Systems*. Sous la dir. d'Albert Y. Zomaya et Young Choon Lee. ISBN: 978-0-470-90875-4. Wiley, 2012, p. 163–187.

[Ch6] Davide Careglio, Georges Da Costa et Sergio Ricciardi. "Hardware leverages for energy reduction in large scale distributed systems". In : *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*. Sous la dir. de Jean-Marc Pierson. ISBN: 978-1-118-86463-0. John Wiley et Sons, 2015. Chap. 2, p. 17–40.

[Ch7] Georges Da Costa, Helmut Hlavacs, Karin Hummel et Jean-Marc Pierson. "Modeling the Energy Consumption of Distributed Applications". In : *Handbook of Energy-Aware and Green Computing*. Sous la dir. d'Ishfaq Ahmad et Sanjay Ranka. ISBN 9781466501164. Chapman & Hall, CRC Press, 2012. Chap. 29, p. 0–0.

[Ch8] Laurent Lefèvre, Sebastien Varrette, Frédéric Pinel, Pascal Bouvry, Ghislain Landry Tsafack Chetsa, Georges Da Costa, Patricia Stolf, Jean-Marc Pierson et Emmanuel Jeannot. "Energy Efficiency and High Performance Computing". In : *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*. Sous la dir. de Jean-Marc Pierson. ISBN: 978-1-118-86463-0. John Wiley et Sons, 2015. Chap. 7, p. 197–224.

[Ch9] Jason Mair, Zhiyi Huang, David Eyers, Leandro Fontoura Cupertino, Georges Da Costa, Jean-Marc Pierson et Helmut Hlavacs. "Power Modeling". In : *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*. Sous la dir. de Jean-Marc Pierson. ISBN: 978-1-118-86463-0. John Wiley et Sons, 2015. Chap. 5, p. 131–158.

[Ch10] Micha Vor Dem Berge et al. "CoolEmAll : Models and Tools for Planning and Operating Energy Efficient Data Centres". In : *Handbook on Data Centers*. Sous la dir. de Samee U. Khan et Albert Y. Zomaya. ISBN: 978-1-4939-2091-4. Springer, 2015, p. 191–245.

Rapports techniques

[RT1] Oleksiak Ariel, Piatek Wojciech, Piontek Tomasz, Grabowski Piotr, Sun Hongyang, Montanera Enric Pere Pages, vor dem Berge Micha, Volk Eugen et Sandoval Yosandra. *D4.8 Final release of the Module Operation Platform*. Rapp. tech. CoolEmAll, 2014.

[RT2] Georges Da Costa, Davide Careglio, Ronen I Kat, Avi Mendelson, Jean-Marc Pierson et Yiannakis Sazeides. *Hardware leverages for energy reduction in large scale distributed systems*. Rapp. tech. IRIT/RT-2010-2-FR. Université Paul Sabatier, Toulouse : Cost IC0804, 2012. url : `http://www.irit.fr/~Georges.Da-Costa/brochure.pdf`.

[RT3] Rathgeb Daniel, Volk Eugen, Sandoval Yosandra, da Costa Georges, Zilio Thomas, vor dem Berge Micha et Piatek Wojciech. *D2.4 First release of the simulation and visualisation toolkit*. Rapp. tech. CoolEmAll, 2013.

[RT4] Volk Eugen, Piątek Wojtek, Da Costa Georges, Jarus Mateusz, vor dem Berge Micha et Sisó Laura. *D2.3 - First definition of the hardware and software models*. Rapp. tech. CoolEmAll, 2013.

[RT5] Volk Eugen, Sandoval Yosandra, da Costa Georges, Zilio Thomas, vor dem Berge Micha, Michels Dirk, Piatek Wojciech et Grabowski Piotr. *D2.5 Second release of the simulation and visualisation toolkit*. Rapp. tech. CoolEmAll, 2014.

[RT6] Da Costa Georges, Jarus Mateusz et Zilio Thomas. *D5.5 Energy- and Heat-aware benchmarks*. Rapp. tech. CoolEmAll, 2013.

[RT7] Luis Prieto Juan, Da Costa Georges, Oleksiak Ariel et Jarus Mateusz. *D5.4 Energy and Heat-aware classification of application*. Rapp. tech. CoolEmAll, 2013.

[RT8] Sisó Laura, Salom Jaume, Oro Eduard, Da Costa Georges et Zilio Thomas. *D5.6 Final metrics and benchmarks*. Rapp. tech. CoolEmAll, 2014.

[RT9] Fontoura Cupertino Leandro et Da Costa Georges. *D5.2 A command-line tool displaying power and energy consumption of each process*. Rapp. tech. CoolEmAll, 2012.

[RT10] vor dem Berge Micha, Christmann Wolfgang, Volk Eugen, Napolitano Assunta, Pierson Jean-Marc, Thiebolt François et Da Costa Georges. *D3.1 First definition of the flexible rack-level compute box with integrated cooling*. Rapp. tech. CoolEmAll, 2012.

[RT11] B. Fornós Ramon, Napolitano Assunta, Salom Jaume, Da Costa Georges et Eugen Volk. *D5.1 White paper on Energy- and Heat-aware metrics for computing modules*. Rapp. tech. CoolEmAll, 2012.

[RT12] Milagros Rey Porto et al. *Data Centres Sustainability Cluster Activities Task 3*. anglais. Rapport de recherche 3. European Commission, 2014. url : `https://ec.europa.eu/digital-agenda/en/news/cluster-fp7-projects-proposes-new-environmental-efficiency-metrics-data-centres`.

[RT13] Zilio Thomas et Da Costa Georges. *D5.3 Energy, heat and communication pattern plug-ins for Ganglia*. Rapp. tech. CoolEmAll, 2013.

[RT14] Piontek Tomasz et al. *D4.1 Architecture of the Module Operation Platform*. Rapp. tech. CoolEmAll, 2012.

[RT15] Wössner Uwe, Volk Eugen, Gallizo Georgina, Piatek Wojciech, Da Costa Georges, Pierson Jean-Marc et Domagalski Piotr. *D2.2 Design of the CoolEmAll simulation and visualisation environment*. Rapp. tech. CoolEmAll, 2012.

Doctorats encadrés ayant soutenus

[D1] Damien Borgetto. "Allocation et Réallocation de Services pour les Économies d'Énergie dans les Clusters et les Clouds". Thèse. Université Toulouse 3 Paul Sabatier, 2013. url : `http://thesesups.ups-tlse.fr/2100/`.

[D2] Leandro Fontoura Cupertino. "Modeling the power consumption of computing systems and applications through Machine Learning techniques". Thèse. Université Toulouse 3 Paul Sabatier, 2015.

[D3] Tom Guérout. "Scheduling under serivce quality constraints in the clouds". Thèse. INSA Toulouse, déc. 2014. url : `https://tel.archives-ouvertes.fr/tel-01110420`.

[D4] Cheikhou Thiam. "Anti Load-Balancing for Energy-Aware Distributed Scheduling of Virtual Machines". Thèse. Université Toulouse 3 Paul Sabatier, 2014. url : `http://thesesups.ups-tlse.fr/2441/`.

Algorithm ∞ Choco-marrons

Require: 200g de chocolat, 500g crème de marrons, 3 oeufs, 100g de beurre, 75g de farine

1: Mélanger Chocolat et beurre
2: **repeat**
3: Chauffer au bain marie
4: Mélanger
5: **until** Mélange soyeux
6: Rajouter Crème de marrons
7: Mélanger
8: Rajouter Oeuf et Farine
9: Mettre dans un moule à *muffins*
10: Mettre au four préchauffé à 150° pendant 25 minutes
11: Déguster

Bibliographie

[1] Cristiana Amza, Anupam Chanda, Alan L Cox, Sameh Elnikety, Romer Gil, Karthick Rajamani, Willy Zwaenepoel, Emmanuel Cecchet et Julie Marguerite. "Specification and implementation of dynamic web site benchmarks". In : *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on*. IEEE. 2002, p. 3–13.

[2] Marc André. *Splendor*. Edité par Space Cowboys. URL : `http://www.trictrac.net/jeu-de-societe/splendor/infos`.

[3] Victor Avelar, Dan Azevedo et Alan French. *PUE : A Comprehensive Examination of the Metric*. Rapp. tech. 49. GreenGrid, 2012.

[4] Luiz André Barroso et Urs Hölzle. "The Case for Energy-Proportional Computing". In : *IEEE Computer* 40 (2007). URL : `http://www.computer.org/portal/site/computer/index.jsp?pageID=computer_level1&path=computer/homepage/Dec07&file=feature.xml&xsl=article.xsl`.

[5] Wolfgang Barth. *Nagios : System and network monitoring*. No Starch Press, 2008.

[6] R. Basmadjian et H. De Meer. "Evaluating and modeling power consumption of multi-core processors". In : *Future Energy Systems : Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*. 2012, p. 1–10.

[7] Robert Basmadjian, Hermann De Meer, Ricardo Lent et Giovanni Giuliani. "Cloud computing and its interest in saving energy : the use case of a private cloud". In : *Journal of Cloud Computing* 1.1 (2012), p. 1–25.

[8] Charles H Bennett. "Notes on Landauer's principle, reversible computation, and Maxwell's Demon". In : *Studies In History and Philosophy of Science Part B : Studies In History and Philosophy of Modern Physics* 34.3 (2003), p. 501–510.

[9] Micha vor dem Berge. *D3.6 Final release of the rack-level and the modular compute boxes*. Rapp. tech. CoolEmAll, 2014.

[10] B Beverly Yang et Hector Garcia-Molina. "Designing a super-peer network". In : *Data Engineering, 2003. Proceedings. 19th International Conference on*. IEEE. 2003, p. 49–60.

[11] Aruna Prem Bianzino, Claude Chaudet, Dario Rossi et J Rougier. "A survey of green networking research". In : *Communications Surveys & Tutorials, IEEE* 14.1 (2012), p. 3–20.

[12] Kurt Binder, Jürgen Horbach, Walter Kob, Wolfgang Paul et Fathollah Varnik. "Molecular dynamics simulations". In : *Journal of Physics : Condensed Matter* 16.5 (2004), S429.

[13] Jacek Błażewicz, Maciej Machowiak, Jan Weglarz, Mikhail Y Kovalyov et Denis Trystram. "Scheduling malleable tasks on parallel processors to minimize the makespan". In : *Annals of Operations Research* 129.1-4 (2004), p. 65–80.

[14] Raphaël Bolze et al. "Grid'5000 : A Large Scale And Highly Reconfigurable Experimental Grid Testbed". In : *International Journal of High Performance Computing Applications* 20.4 (2006), p. 481–494. doi : 10.1177/1094342006070078. url : https://hal.inria.fr/hal-00684943.

[15] D. Brooks, V. Tiwari et M. Martonosi. "Wattch : a framework for architectural-level power analysis and optimizations". In : *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*. 2000, p. 83–94.

[16] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault et Raymond Namyst. "hwloc : A generic framework for managing hardware affinities in HPC applications". In : *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*. IEEE. 2010, p. 180–186.

[17] Laurent Broto, Daniel Hagimont, Patricia Stolf, Noel Depalma et Suzy Temate. "Autonomic management policy specification in tune". In : *Proceedings of the 2008 ACM symposium on Applied computing*. ACM. 2008, p. 1658–1663.

[18] Richard Brown et al. "Report to congress on server and data center energy efficiency : Public law 109-431". In : *Lawrence Berkeley National Laboratory* (2008).

[19] Rajkumar Buyya, Anton Beloglazov et Jemal Abawajy. "Energy-efficient management of data center resources for cloud computing : A vision, architectural elements, and open challenges". In : *arXiv preprint arXiv :1006.0308* (2010).

[20] Rajkumar Buyya et Manzur Murshed. "Gridsim : A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing". In : *Concurrency and Computation : Practice and Experience*. T. 14. Wiley Online Library, 2002, p. 1175–1220.

[21] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose et Rajkumar Buyya. "CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". In : *Software : Practice and Experience* 41.1 (2011), p. 23–50.

[22] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson et Frédéric Suter. "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms". In : *Journal of Parallel and Distributed Computing* 74.10 (juin 2014), p. 2899–2917. url : `http://hal.inria.fr/hal-01017319`.

[23] Roberto G. Cascella, Christine Morin, Jean-Pierre Banâtre et Thierry Priol. *Private-by-Design : Towards Personal Local Clouds*. Research Report RR-8634. Inria Rennes ; INRIA, 2014.

[24] Bruno Cathala et Charles Chevallier. *Abyss*. Edité par Bombyx. Illustration par Xavier Collette. url : `http://www.trictrac.net/jeu-de-societe/abyss/infos`.

[25] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen et Yan Chen. "OSA : an optical switching architecture for data center networks with unprecedented flexibility". In : *IEEE/ACM Transactions on Networking (TON)* 22.2 (2014), p. 498–511.

[26] Ghislain Landry Tsafack Chetsa. "System Profiling and Green Capabilities for Large Scale and Distributed Infrastructures". Thèse de doct. Ecole normale supérieure de lyon-ENS LYON, 2013.

[27] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt et Andrew Warfield. "Live migration of virtual machines". In : *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, p. 273–286.

[28] Bailey David, Harris Tim, Saphir William, van der Wijngaart Rob, Woo Alex et Yarrow Maurice. *The NAS Parallel Benchmarks 2.0*. Rapp. tech. NAS Technical Report NAS-95-020 NASA Ames Research Center, Moffett Field CA, 1995.

[29] T. Do, S. Rawshdeh et W. Shi. "pTop : A Process-level Power Profiling Tool". In : *2nd Workshop on Power Aware Computing and Systems (HotPower'09)*. 2009.

[30] Jack Dongarra et Piotr Luszczek. "Linpack benchmark". In : *Encyclopedia of Parallel Computing*. Springer, 2011, p. 1033–1036.

[31] Jack J Dongarra, Hans W Meuer et Erich Strohmaier. *Top500 supercomputer sites*. 1994.

[32] Truong Vinh Truong Duy, Yukinori Sato et Yasushi Inoguchi. "Performance evaluation of a green scheduling algorithm for energy savings in cloud computing". In : *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE. 2010, p. 1–8.

[33] Khosrow Ebrahimi, Gerard F Jones et Amy S Fleischer. "A review of data center cooling technology, operating conditions and the corresponding low-grade waste heat recovery opportunities". In : *Renewable and Sustainable Energy Reviews* 31 (2014), p. 622–638.

[34] D. Economou, S. Rivoire et C. Kozyrakis. "Full-system power analysis and modeling for server environments". In : *In Workshop on Modeling Benchmarking and Simulation (MOBS*. 2006.

[35] Emerson. *Data Center Users Group : Survey Results*. Rapp. tech. Emerson Network Power, Emerson Electric Co., 2014.

[36] Emerson. *Energy logic 2.0 : New strategies for cutting data center energy costs and boosting capacity*. Rapp. tech. Emerson Network Power, Emerson Electric Co., 2014.

[37] Biscotti Fabrizio, Skorupa Joe, Contu Ruggero, Tratz-Ryan Bettina, Rasit Errol, Lerner Andrew, Kros Anthony et Zhang Jie. *The Impact of the Internet of Things on Data Centers*. Rapp. tech. G00250562. Gartner, 2014.

[38] Eugen Feller, Louis Rilling, Christine Morin, Renaud Lottiaux et Daniel Leprince. "Snooze : a scalable, fault-tolerant and distributed consolidation manager for large-scale clusters". In : *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*. IEEE Computer Society. 2010, p. 125–132.

[39] J Fontán, T Vázquez, L Gonzalez, Ruben S Montero et IM Llorente. "OpenNEbula : The open source virtual machine manager for cluster computing". In : *Open Source Grid and Cluster Software Conference, San Francisco, CA, USA*. 2008.

[40] Lance Fortnow. "The status of the P versus NP problem". In : *Communications of the ACM* 52.9 (2009), p. 78–86.

[41] Ian Foster et Carl Kesselman. *The Grid 2 : Blueprint for a new computing infrastructure*. Elsevier, 2003.

[42] Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol, Helio Tejedor et Robert Rallo. "Planetsim : A new overlay network simulation framework". In : *Software engineering and middleware*. Springer, 2005, p. 123–136.

[43] Cook Gary. "Clicking Clean : How Companies are Creating the Green Internet". In : *Greenpeace International* (2014).

[44] Cook Gary et Van Horn Jodie. "How dirty is your data ?" In : *Greenpeace International* (2011).

[45] Brett J. Gilbert et Matthew Dunstan. *Elysium*. Edité par Space Cowboys. url : http://www.trictrac.net/jeu-de-societe/elysium-0/infos.

[46] Zhenhuan Gong, Xiaohui Gu et John Wilkes. "Press : Predictive elastic resource scaling for cloud systems". In : *Network and Service Management (CNSM), 2010 International Conference on*. IEEE. 2010, p. 9–16.

[47] Make IT Green. "Cloud Computing and its Contribution to Climate Change". In : *Greenpeace International* (2010).

[48] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner et Amin Vahdat. "Enforcing performance isolation across virtual machines in Xen". In : *Middleware 2006*. Springer, 2006, p. 342–362.

[49] Isabelle Guyon et André Elisseeff. "An introduction to variable and feature selection". In : *The Journal of Machine Learning Research* 3 (2003), p. 1157–1182.

[50] Chen H, Li Y et Shi W. "Fine-grained power management using process-level profiling". In : *Sustainable Computing : Informatics and Systems* 2.1 (2012), p. 33–42. issn : 2210–5379. doi : http://dx.doi.org/10.1016/j.suscom.2012.01.002. url : http://www.sciencedirect.com/science/article/pii/S2210537912000030.

[51] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller et Julia Lawall. "Entropy : a consolidation manager for clusters". In : *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM. 2009, p. 41–50.

[52] Edgar G Hertwich. "Consumption and the rebound effect : an industrial ecology perspective". In : *Journal of Industrial Ecology* 9.1-2 (2005), p. 85–98.

[53] Christina Hoffa, Gaurang Mehta, Timothy Freeman, Ewa Deelman, Kate Keahey, Bruce Berriman et John Good. "On the use of cloud computing for scientific workflows". In : *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE. 2008, p. 640–645.

[54] Ye Huang, Amos Brocco, Michele Courant, Beat Hirsbrunner et Pierre Kuonen. "MaGate simulator : a simulation environment for a decentralized grid scheduler". In : *Advanced Parallel Processing Technologies*. Springer, 2009, p. 273–287.

[55] Sadeka Islam, Jacky Keung, Kevin Lee et Anna Liu. "Empirical prediction models for adaptive resource provisioning in the cloud". In : *Future Generation Computer Systems* 28.1 (2012), p. 155–162.

[56] Raj Jain. *The art of computer systems performance analysis*. John Wiley & Sons, 2008.

[57] Jungsoo Kim, Mohamed M Sabry, David Atienza, Kalyan Vaidyanathan et Kenny Gross. "Global fan speed control considering non-ideal temperature measurements in enterprise servers". In : *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE. 2014, p. 1–6.

[58] Dalibor Klusáček et Hana Rudová. "Alea 3 : job scheduling simulator". In : *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics et Telecommunications Engineering). 2010, p. 61.

[59] Yuetsu Kodama, Satoshi Itoh, Toshiyuki Shimizu, Satoshi Sekiguchi, Hiroshi Nakamura et Naohiko Mori. "Imbalance of CPU temperatures in a blade system and its impact for power consumption of fans". In : *Cluster Computing* 16.1 (2013), p. 27–37.

[60] Krzysztof Kurowski, Ariel Oleksiak, W Piatek, Tomasz Piontek, A Przybyszewski et J Weglarz. "DCworms–A tool for simulation of energy efficiency in distributed computing infrastructures". In : *Simulation Modelling Practice and Theory* 39 (2013), p. 135–151.

[61] Rolf Landauer. "Irreversibility and heat generation in the computing process". In : *IBM journal of research and development* 5.3 (1961), p. 183–191.

[62] Klaus-Dieter Lange. "Identifying Shades of Green : The SPECpower Benchmarks." In : *IEEE Computer* 42.3 (2009), p. 95–97.

[63] Eugene L Lawler et David E Wood. "Branch-and-bound methods : A survey". In : *Operations research* 14.4 (1966), p. 699–719.

[64] Jie Liu, Michel Goraczko, Sean James, Christian Belady, Jiakang Lu et Kamin Whitehouse. "The data furnace : heating up with cloud computing". In : *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing, HotCloud*. T. 11. 2011, p. 15–15.

[65] Charles Lively, Xingfu Wu, Valerie Taylor, Shirley Moore, Hung-Ching Chang et Kirk Cameron. "Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems". In : *Int. J. High Perform. Comput. Appl.* 25.3 (août 2011), p. 342–350. issn : 1094-3420. doi : 10.1177/1094342011414749. url : http://dx.doi.org/10.1177/1094342011414749.

[66] Arlitt M. et Jin T. *World Cup Web Site Access Logs*. 1998. url : http://www.acm.org/sigcomm/ITA/.

[67] Andrew Makhorin. *GLPK (GNU linear programming kit)*. 2008. url : http://www.gnu.org/software/glpk/.

[68] Matthew L Massie, Brent N Chun et David E Culler. "The ganglia distributed monitoring system : design, implementation, and experience". In : *Parallel Computing* 30.7 (2004), p. 817–840.

[69] Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane, Demetrios Dimatos, Gary Hamilton, Michael McCabe et James Owens. "Quantifying the performance isolation properties of virtualization systems". In : *Proceedings of the 2007 workshop on Experimental computer science*. ACM. 2007, p. 6.

[70] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren et R. K. Gupta. "Evaluating the Effectiveness of Model-based Power Characterization". In : *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC'11. Portland, OR : USENIX Association, 2011, p. 12–12. URL : `http://dl.acm.org/citation.cfm?id=2002181.2002193`.

[71] Dejan S Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins et Zhichen Xu. *Peer-to-peer computing*. 2002.

[72] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[73] Alberto Montresor et Márk Jelasity. "PeerSim : A scalable P2P simulator". In : *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*. IEEE. 2009, p. 99–100.

[74] Gordon E Moore. "Lithography and the future of Moore's law". In : *SPIE's 1995 Symposium on Microlithography*. International Society for Optics et Photonics. 1995, p. 2–17.

[75] Justin D Moore, Jeffrey S Chase, Parthasarathy Ranganathan et Ratnesh K Sharma. "Making Scheduling" Cool" : Temperature-Aware Workload Placement in Data Centers." In : *USENIX annual technical conference, General Track*. 2005, p. 61–75.

[76] A. Noureddine, A. Bourdon, R. Rouvoy et L. Seinturier. "A preliminary study of the impact of software engineering on GreenIT". In : *Green and Sustainable Software (GREENS), 2012 First International Workshop on*. 2012, p. 21–27. DOI : `10.1109/GREENS.2012.6224251`.

[77] Daniel Nurmi, Richard Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff et Dmitrii Zagorodnov. "The eucalyptus open-source cloud-computing system". In : *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*. IEEE. 2009, p. 124–131.

[78] *OpenStack Home Page*. URL : `http://www.openstack.org/`.

[79] Vilfredo Pareto. *Manual of political economy*. Macmillan, Co, 1927.

[80] Dana Petcu. "Portability and interoperability between clouds : challenges and case study". In : *Towards a Service-Based Internet*. Springer, 2011, p. 62–74.

[81] Cris Pettey. “Gartner estimates ICT industry accounts for 2 percent of global CO2 emissions”. In : *https://www.gartner.com/newsroom/id/503867* 14 (2007), p. 2013.

[82] Eduardo Pinheiro, Wolf-Dietrich Weber et Luiz André Barroso. “Failure Trends in a Large Disk Drive Population.” In : *FAST*. T. 7. 2007, p. 17–23.

[83] Flavien Quesnel et Adrien Lèbre. “Cooperative dynamic scheduling of virtual machines in distributed systems”. In : *Euro-Par 2011 : Parallel Processing Workshops*. Springer. 2012, p. 457–466.

[84] Nikola Rajovic, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones et Alex Ramirez. “Tibidabo : Making the case for an ARM-based HPC system”. In : *Future Generation Computer Systems* 36 (2014), p. 322–334.

[85] Alex Ramirez et Paul Carpenter. “European scalable and power efficient HPC platform based on low-power embedded technology”. In : *Montblanc Project* (2013).

[86] Villars Richard L. et Shirer Michael. *Worldwide Datacenter Census and Construction 2014–2018 Forecast : Aging Enterprise Datacenters and the Accelerating Service Provider Buildout*. Rapp. tech. 251830. IDC, 2014.

[87] Suzanne Rivoire, Parthasarathy Ranganathan et Christos Kozyrakis. “A Comparison of High-Level Full-System Power Models.” In : *HotPower* 8 (2008), p. 3–3.

[88] François Rossigneux, Jean-Patrick Gelas, Laurent Lefevre et Marcos Dias De Assuncao. “A Generic and Extensible Framework for Monitoring Energy Consumption of OpenStack Clouds”. In : *arXiv preprint arXiv :1408.6328* (2014).

[89] E. Rotem, A. Naveh, A. Ananthakrishnan, D. Rajwan et E. Weissmann. “Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge”. In : *IEEE Micro* 32.2 (2012), p. 20–27. issn : 0272-1732. doi : http://doi.ieeecomputersociety.org/10.1109/MM.2012.12.

[90] Antonio G Ruzzelli, C Nicolas, Anthony Schoofs et Gregory MP O’Hare. “Real-time recognition and profiling of appliances through a single electricity sensor”. In : *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*. IEEE. 2010, p. 1–9.

[91] Jyotiprakash Sahoo, Subasish Mohapatra et Radha Lath. “Virtualization : A survey on concepts, taxonomy and associated security issues”. In : *Computer and Network Technology (ICCNT), 2010 Second International Conference on*. IEEE. 2010, p. 222–226.

[92] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, W. Wang et J. G. Powers. “A description of the advanced research WRF version 2”. In : *NCAR Tech Note* NCAR/TN-468+STR (2005).

[93] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente et Ian Foster. "Virtual infrastructure management in private and hybrid clouds". In : *Internet Computing, IEEE* 13.5 (2009), p. 14–22.

[94] Daniel Spielman et Shang-Hua Teng. "Smoothed analysis of algorithms : Why the simplex algorithm usually takes polynomial time". In : *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. ACM. 2001, p. 296–305.

[95] Matt Stansberry. *2014 Data Center Industry Survey*. Rapp. tech. Uptime Institute, 2014.

[96] Lindenberg Steve, Smith Brian, Kathy O'Dell, DeMeo Ed et Ram Bonnie. *20% Wind Energy by 2030 : Increasing Wind Energy's Contribution to U.S. Electricity Supply*. Rapp. tech. DOE/GO-102008-2567. U.S. Department of Energy (DOE), 2008.

[97] Mark Stillwell, David Schanzenbach, Frédéric Vivien et Henri Casanova. "Resource allocation algorithms for virtualized service hosting platforms". In : *Journal of Parallel and Distributed Computing* 70.9 (2010), p. 962–974.

[98] Andrew Tanenbaum et Maarten Van Steen. *Distributed systems*. Pearson Prentice Hall, 2007.

[99] Qinghui Tang, Sandeep KS Gupta et Georgios Varsamopoulos. "Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers : A cyber-physical approach". In : *Parallel and Distributed Systems, IEEE Transactions on* 19.11 (2008), p. 1458–1472.

[100] Qinghui Tang, Tridib Mukherjee, Sandeep KS Gupta et Phil Cayton. "Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters". In : *Intelligent Sensing and Information Processing, 2006. ICISIP 2006. Fourth International Conference on*. IEEE. 2006, p. 203–208.

[101] *The grid workloads archive. [Online]*. accessed 2013. url : `http://gwa.ewi.tudelft.nl/pmwiki/`.

[102] Dimitrios Tsoumakos et Nick Roussopoulos. "A Comparison of Peer-to-Peer Search Methods." In : *WebDB*. Citeseer. 2003, p. 61–66.

[103] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres et Maik Lindner. "A break in the clouds : towards a cloud definition". In : *ACM SIGCOMM Computer Communication Review* 39.1 (2008), p. 50–55.

[104] A Vladishev. "Open Source Enterprise Monitoring with Zabbix". In : *Open Source Data Center Conference, Nurnberg*. T. 60. 2009.

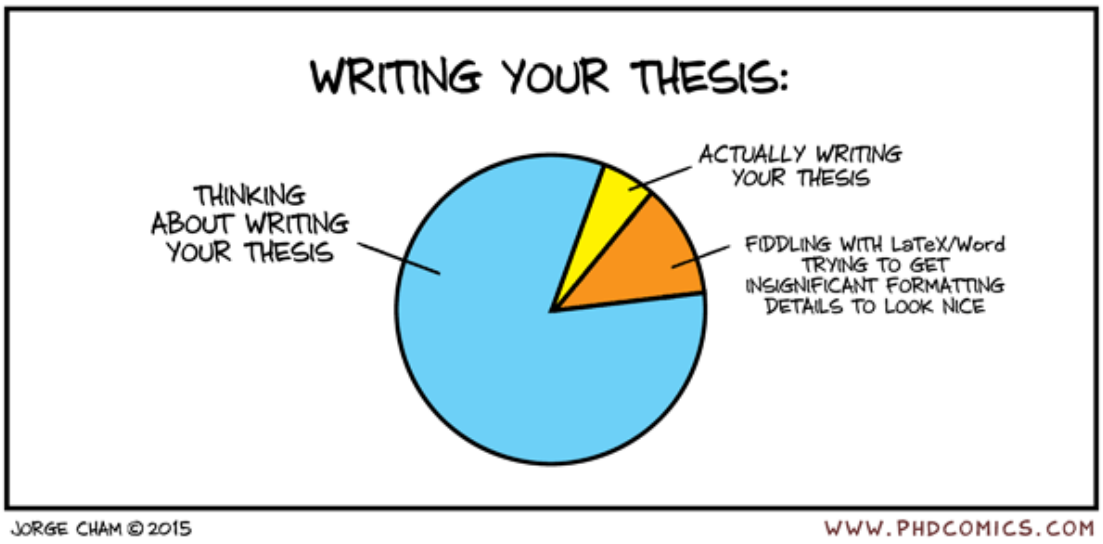
[105] Michael White et al. *Monty Python and the Holy Grail*. Columbia TriStar Home Entertainment. 2001.

[106] Ryan Wiser et Mark Bolinger. "2013 Wind technologies market report". In : *U.S. Department of energy* (2014).

[107] Laurence A Wolsey. “Heuristic analysis, linear programming and branch and bound”. In : *Combinatorial Optimization II*. Springer, 1980, p. 121–134.

[108] Jun Yang, Xiuyi Zhou, Marek Chrobak, Youtao Zhang et Lingling Jin. “Dynamic thermal management through task scheduling”. In : *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on*. IEEE. 2008, p. 191–201.

[109] Yagiz Onat Yazir, Chris Matthews, Roozbeh Farahbod, Stephen Neville, Adel Guitouni, Sudhakar Ganti et Yvonne Coady. “Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis”. In : *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. Ieee. 2010, p. 91–98.



Piled Higher and Deeper by Jorge Cham www.phdcomics.com