La gestion des ressources pour des infrastructures vertes par la reconfiguration

Patricia Stolf

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 *13/11/2015* **par :**

PATRICIA STOLF

La gestion des ressources pour des infrastructures vertes par la reconfiguration

---

JURY

FRÉDÉRIC DESPREZ	Directeur de recherche	Rapporteur
JEAN-MARC MENAUD	Professeur d'Université	Rapporteur
LIONEL SEINTURIER	Professeur d'Université	Rapporteur
PASCAL BOUVRY	Professeur d'Université	Membre du Jury
LAURENT LEFÈVRE	Chargé de recherche, HDR	Membre du Jury
NOUREDINE MELAB	Professeur d'Université	Membre du Jury
THIERRY MONTEIL	Maître de Conférences, HDR	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)

Remerciements

Mes remerciements s'adressent tout d'abord à Jean-Marc Pierson responsable de l'équipe SEPIA dans laquelle j'effectue mes recherches. Sa confiance, ses compétences et ses qualités humaines sont importantes au quotidien et contribuent à rendre les conditions de travail agréables. Nos échanges à l'occasion de la rédaction de ce manuscrit ont été précieux.

Ensuite, je souhaiterais remercier mes directeurs de thèse : Thierry Monteil et Gérard Authié. C'est grâce à eux que j'exerce le métier d'enseignant chercheur. J'ai depuis co-encadré deux doctorants avec Thierry Monteil et c'est toujours un plaisir.

Je suis très reconnaissante envers mes rapporteurs et les membres de mon jury pour me faire l'honneur qu'il me font de participer à mon jury.

Ses travaux n'auraient pu avoir lieu sans les collaborations avec les collègues de projet, membres de l'équipe (Daniel Hagimont, Laurent Broto, Georges Da Costa et bien sûr Jean-Marc Pierson), les doctorants et post-doctorants que j'ai encadrés : Rémi Sharrock, Landry Tsafack, Cédric Eichler, Violaine Villebonnet, Damien Borgetto, Hongyang Sun.

La collaboration avec Laurent Lefèvre au cours de deux thèses en commun est pour moi très importante, nos échanges réguliers en visio sont toujours intéressants et dans la bonne humeur !

Je profite de cet avant propos pour saluer les membres du laboratoire avec lesquels j'ai travaillé, mon "co-bureau" Geo qui sait créer les moments de convivialité nécessaires !

Merci à toutes les personnes que j'ai pu côtoyer durant ces années et avec lesquelles j'ai travaillé avec plaisir. Une pensée pour mes collègues de l'IUT de Blagnac qui m'ont prodigué conseils, soutien...

Enfin, mon dernier mot sera pour mes proches qui ont subi une vie de famille un peu perturbée, je pense particulièrement à mes enfants qui ont du supporter une maman qui devait "travailler et faire de l'ordinateur".

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte de la recherche et motivations

1.1.1 Les systèmes informatiques

Ce mémoire se situe dans le contexte des systèmes informatiques à grande échelle pouvant être des grilles de calcul ou le cloud computing. Nous pouvons faire deux constats sur ces systèmes : la consommation énergétique de ces systèmes est trop importante ; ces systèmes sont de plus en plus complexes et distribués.
Les utilisateurs demandent toujours plus de performance de calcul ou toujours plus de stockage de données et toujours plus de disponibilité des services. Les centres de calcul sont maintenant composés de plusieurs milliers de machines constituant ainsi des systèmes large-échelle. Ils voient leur utilisation croître, en raison de la performance des ressources et grâce à la transparence dans l'utilisation des services offerts via le cloud. Ainsi, cette course à la puissance a conduit les infrastructures à des performances allant du *TeraFlop* en 1997, au *PetaFlop* en 2008 et *l'ExaFlop* est annoncé pour 2020. La course à la disponibilité a conduit à des infrastructures dont les ressources sont sur-provisionnées afin de pouvoir répondre aux pic de charge.

Un datacenter héberge des ordinateurs interconnectés, fonctionne jour et nuit sans interruption, produit des données, en renvoie ; c'est une vraie usine numérique. Associé au flux des données numériques, il y a le flux électrique et le flux d'air frais pour refroidir l'infrastructure. Tout ceci n'est pas sans conséquence sur la consommation énergétique.

En parallèle, l'architecture des systèmes informatiques et les logiciels deviennent de plus en plus complexes et distribués. La taille et la complexité d'un tel système empêchent une gestion efficace par des administrateurs système sans erreurs et rapide. IBM a introduit le

concept d'informatique autonomique en 2001 [KC03]. Ce concept est inspiré de la biologie, le corps humain s'auto-régule (rythme cardiaque, pression sanguine ...) en fonction des efforts à fournir et des événements extérieurs. Le but de l'informatique autonomique est de surpasser la complexité des systèmes logiciels et matériels en leur procurant des capacités d'auto-gestion. Le système devient conscient des événements se produisant et y réagit de façon autonome et adaptée. On peut imaginer exploiter ce concept pour réduire la facture énergétique. L'utilisation de méthodes collaboratives et l'informatique autonomique pourrait résoudre beaucoup de problèmes : configuration des applications, performance à travers des reconfigurations auto-gérées par le système. Il est alors intéressant d'utiliser et appliquer la boucle autonomique pour intégrer une gestion efficace et autonome des ressources : optimisation de la performance, optimisation de la consommation énergétique, pour réagir, adapter le comportement des infrastructures matérielles et logicielles tout en respectant l'objectif global à atteindre.

1.1.2 Quelques chiffres

La consommation énergétique des infrastructures des centres de calcul, de données, des grilles ou cloud est un challenge important. Les centres de données consomment 2% de la consommation d'électricité mondiale, ce qui représente 200 milliards de kWh [Koo11]. La consommation d'un data center de 10 000 mètres carrés est approximativement égale à celle d'une ville de 50 000 habitants.
En 2007, l'électricité demandée conjointement par internet et le cloud se serait classée 5ème au niveau mondial parmi les demandes de consommation des pays ; ainsi si l'on avait considéré le Cloud Computing comme un pays, il aurait été le 5ème pays le plus consommateur d'électricité [how].

Le processeur a historiquement dominé la consommation des systèmes, toutefois depuis 2010, on observe une inversion de la tendance avec une consommation de la mémoire, des disques et des communications qui peuvent représenter jusqu'à 55% de la consommation [LZ10]. Au niveau d'un centre de données, la consommation énergétique a différentes origines : les serveurs consomment 45%, le système de refroidissement 25%, le réseau 15% et 15% de l'énergie est gaspillée. [PMWV09] présente une description complète.

Sachant que la consommation à vide d'un processeur est comprise entre 80 et 100W alors que chargé elle est comprise entre 200 et 250W [SPE12] ; une gestion efficace des ressources peut permettre des gains dans les consommations énergétiques non négligeables. Ceci ne peut se faire sans considérer l'impact sur la qualité de service pour les utilisateurs.

La consommation d'énergie a un coût économique mais aussi un impact écologique. En 2009, l'Union internationale des télécommunications (ITU) a évalué la contribution du secteur des technologies de l'information et des communications (ICT) au changement climatique entre 2 et 2.5% d'émissions de dioxyde de carbone mondiales [ITU09]. L'empreinte

écologique des data centers ne cesse de croître ; on estime que les besoins énergétiques pourraient doubler d'ici 5 ans. Les industries, les institutions et les gouvernements considèrent de plus en plus l'efficacité énergétique des infrastructures. Il est donc essentiel pour les propriétaires de centres de données et les opérateurs d'évaluer et améliorer la performance en tenant compte du rendement énergétique.

1.1.3 Les métriques

Le GreenGrid [Greb], consortium international visant à améliorer l'efficacité des ressources des data centers, a défini plusieurs métriques dont le PUE (Power Usage Effectiveness). Le PUE est un indicateur qui qualifie l'efficacité énergétique d'un data center en calculant le ratio entre l'énergie totale consommée (incluant le système de refroidissement, l'éclairage ...) par l'ensemble du data center et la partie effectivement consommée par les systèmes informatiques (serveurs, stockage, réseau) que le data center exploite. Plus la valeur du PUE est élevée et moins le centre de calcul est efficace en énergie. Le PUE mondial moyen d'un data center de 2011 à 2014 se situe entre 1,7 (en 2014) et 1,89 (en 2011) [Upt] alors que l'efficacité énergétique idéale serait de 1.

Le PUE est une métrique utilisée mais aussi critiquée : elle est considérée par certains comme injuste car elle mesure l'efficacité énergétique au niveau du datacenter et non au niveau des équipements. Différents travaux ont proposé des métriques pour évaluer l'efficacité énergétique et l'émission de gaz à effet de serre [Greb],[SSJ+13],[VTG+13].
D'autres métriques évaluent la performance par watt : la liste *Green500* [Grea] évalue les *FLOPS* (Floating Point Operations Per Second) par watt des différents systèmes évalués.

1.1.4 Les solutions

La consommation de puissance a longtemps été étudiée pour les équipements mobiles : portables ou tablettes dans l'objectif de prolonger la durée des batteries. Dans le domaine des datacenters, c'est une préoccupation plus récente suite à deux motivations : tout d'abord la volonté de réduire les coûts d'alimentation et de refroidissement mais aussi grâce à une prise de conscience de l'impact environnemental.

Le changement de matériel des datacenters par du matériel plus récent et plus efficace énergétiquement est possible grâce aux efforts matériels des constructeurs mais cela ne résout pas le problème du sur-approvisionnement en ressources. Les efforts matériels doivent être complétés par des approches logicielles.

Globalement trois moyens d'actions sont possibles : en rendant les applications efficaces, en proposant une gestion des ressources *energy aware* et en rendant le matériel plus efficace

[VLK+13]. Dans ce mémoire nous proposons de corréler les deux premiers.

1.2 Problématique et axes de recherche

Ce mémoire se propose de répondre à la problématique suivante :
comment gérer de manière optimale les ressources afin d'obtenir des infrastructures matérielles et logicielles *vertes* c'est à dire efficace en énergie ?
Nous chercherons quels leviers verts nous pouvons utiliser, quelles reconfigurations autonomiques nous pouvons mettre en oeuvre afin d'atteindre cet objectif. Pour finir, nous nous proposons de décrire ce que pourrait être un centre de décision autonome pour des infrastructures *vertes*. Un tel centre de décision autonome aurait comme mission une bonne gestion des ressources et ce pour gagner en efficacité (énergie, thermique mais aussi optimisation de ressources au sens large).

Pour se faire nous étudions le problème à travers différents axes de recherche :
Tout d'abord dans le premier axe, nous considérons le système dans son ensemble : nous considérons les noeuds d'exécution sans aucune connaissance des applications qui s'exécutent. Nous étudions l'étude de l'efficacité énergétique avec des leviers verts et des reconfigurations bas niveau de l'infrastructure matérielle. Dans ce même axe de recherche nous montrerons comment obtenir une consommation d'énergie proportionnelle avec une infrastructure composée d'architectures hétérogènes en supposant avoir une prédiction du profil de charge des applications.

Puis dans un deuxième axe : nous nous plaçons au niveau *middleware*, au niveau des décisions de placement pour un centre de données ou dans le nuage. Pour cela, nous disposons d'un descriptif du besoin de ressources des applications et une connaissance de l'infrastructure matérielle. Par placement et grâce à des leviers verts de plus haut niveau : la consolidation (migration) et l'arrêt de machines on peut gagner en énergie consommée. Nous considérons plusieurs critères au problème d'optimisation découlant du placement : la performance, la consommation énergétique des serveurs et la consommation du système de refroidissement.

Dans un troisième axe, nous étudions les reconfigurations autonomiques d'applications. A partir de description d'application à base de profil UML ou de graphes, nous pouvons exprimer les propriétés d'auto-gestion des systèmes autonomiques. Nous proposons dans cet axe des reconfigurations de haut-niveau logicielles ou matérielles.

Ce mémoire est structuré de la manière suivante : le chapitre deux est consacré à l'étude de l'efficacité énergétique. Nous présenterons deux approches globales se focalisant sur l'infrastructure. La première approche montrera comment réduire la consommation énergétique dynamique du système complet sans connaissance à priori des applications s'exécutant. La

deuxième présentera une approche innovante visant à réduire les coûts statiques de l'infrastructure en profitant de ressources de calcul (processeurs) hétérogènes.
Le chapitre trois étudiera l'amélioration de l'efficacité énergétique au niveau du placement des applications. Nous présenterons différentes heuristiques de résolution des problèmes multi-critères considérés.
Le chapitre quatre présentera comment les reconfigurations dynamiques de systèmes autonomiques peuvent être utilisées pour renforcer l'efficacité énergétique. Deux approches de formalisation des styles architecturaux des applications et des reconfigurations seront présentées : une approche basée sur UML et une basée sur la théorie des graphes.
Enfin, le chapitre cinq exposera la construction d'un centre de décision autonome et vert.
Pour finir, le chapitre six conclura ce mémoire et des perspectives seront mises à jour.

Chapitre 2

Efficacité énergétique

2.1 Problématique

La consommation énergétique est devenue un facteur limitant pour les infrastructures de type Cloud Computing ou HPC. De nombreuses recherches se sont emparées du problème. Les constructeurs proposent des solutions matérielles avec des composants plus efficaces mais les progrès sont rattrapés par l'utilisation sans cesse croissante de telles infrastructures qui au final sont toujours énergivores. Il est donc nécessaire de proposer des solutions logicielles. Nous avons fait le choix d'étudier le problème en proposant des solutions *en ligne* c'est à dire pendant l'exécution des applications *au runtime* et de considérer l'infrastructure complète, c'est à dire l'ensemble des noeuds de la plate forme. L'objectif est de tendre vers des infrastructures dont la consommation serait proportionnelle à la charge. Les approches logicielles ont peu de succès car elles sont complexes. Elles se focalisent sur l'application et requièrent souvent trop de compétences de la part du programmeur ou de l'utilisateur rendant difficile leur utilisation par des non-experts. Les systèmes HPC sont classés en fonction de leur performance [Top] mais aussi de leur efficacité énergétique [Grea] (c'est à dire du nombre de petaflops par watt) car la course à la performance les a amenés à consommer de plus en plus. Les constructeurs font des efforts mais cela reste insuffisant. Souvent les approches logicielles ne prennent en compte que le processeur. Or le processeur n'est pas le seul composant énergivore. En 2010, les sous-systèmes des infrastructures HPC incluant la mémoire, le stockage et les communications ont consommé 55% de la consommation d'un supercalculateur typique [LZ10]. Tous les sous-sytèmes doivent être considérés : processeur, mémoire, réseau. On a longtemps considéré le processeur comme étant le plus gourmand consommateur de puissance parmi les sous-systèmes HPC. Or ce n'est plus le cas car ils ont adoptés plus de techniques d'efficacité énergétique que les autres composants. Selon Samsung, considérant 8 heures actives et 16 heures d'inactivité pour un serveur, le

sous-système mémoire est responsable de 15% de la consommation d'énergie globale d'un serveur 16GB (technologie DDR2). La consommation de serveurs 32GB et 48GB de même technologie est plus importante et représente 21% et 26% de leur consommation d'énergie totale respective. Dans le dernier cas, la consommation d'énergie de la mémoire excède celle du processeur (20%). L'utilisation des machines dans le contexte HPC (supercalculateurs) et celles des serveurs n'est pas la même, toutefois, nous proposons de gérer leur consommation de manière globale en considérant tous les composants et indépendamment du contexte. Nous visons à détecter les changements d'utilisation de l'infrastructure au cours du temps et à utiliser les leviers verts adaptés.

Barroso et Holzle [BH07] ont pointé en 2007 la nécessité d'avoir des serveurs consommant une énergie proportionnelle à leur charge. C'est à dire une consommation proche de zéro lorsque le système n'est pas utilisé puis une consommation proportionnelle à l'utilisation lorsque le système exécute des applications. Ils ont fait plusieurs constatations : la première est que les serveurs sont rarement utilisés à 100% ; et rarement idle. Ils ont montré que la charge moyenne des machines au sein d'un datacenter de Google était entre 10 et 50%. Si le processeur peut avoir un profil proche du proportionnel ce n'est pas le cas des autres composants.
Avoir un serveur ayant une consommation énergétique proportionnelle permettrait de doubler son efficacité énergétique.

Nous allons dans ce chapitre exposer deux approches de gestion de l'efficacité énergétique au niveau système, au niveau de l'infrastructure. Nous avons étudié l'efficacité énergétique selon deux axes lors de deux thèses dont une est en cours. Dans le premier axe, nous avons cherché à diminuer la consommation dynamique de l'infrastructure en proposant une méthodologie pro-active appliquée au système complet et adaptée pour différents types de systèmes distribués à grande échelle. Selon nous, il est pertinent de détecter l'activité de l'ensemble des applications s'exécutant sur les différents noeuds, de suivre dynamiquement les évolutions et d'utiliser les différents leviers disponibles. Chaque composant (ou presque) de l'infrastructure peut bénéficier d'actions permettant de réduire la consommation énergétique. Nous sommes convaincus qu'il est nécessaire de caractériser le type d'éxécution en cours sur chaque noeud et d'activer les leviers adaptés. Tous les composants matériels doivent être pris en compte : le DVFS (consiste à réduire la fréquence du processeur et donc à réduire la consommation énergétique car la fréquence a un impact quadratique sur l'énergie) est insuffisant car agit uniquement sur le processeur. Ces actions sur le processeur consiste à utiliser différents couples voltage-fréquence variant selon les processeurs (appelés *P-State* pour performance state). Le DVFS n'est utile que sur certaines applications. Dans le cas des applications HPC, applications composées de périodes de calcul et de communications (au sens large : disque, mémoire, réseau), diminuer la fréquence du processeur pendant la période de calcul entrainera un retard de l'exécution et augmentera la consommation énergétique de l'application puisque l'énergie consommée est le produit

de la puissance par le temps. Ce sera au final contre-productif. Par contre, diminuer la fréquence pendant la période de communication ne modifiera pas le temps d'exécution et diminuera la consommation énergétique. Dans [DCP15], les auteurs ont montré que le levier DVFS doit être utilisé au bon moment de l'exécution et ont développé un daemon linux mettant en oeuvre des changements de fréquence sur des benchmarks HPC. Dans l'étude que nous présentons dans ce chapitre, nous avons de manière similaire fait le choix d'utiliser les compteurs de performance pour en déduire le type de comportement du système (utilisation de type calcul intensif, de type mémoire intensif ...). Le comportement des applications pouvant évoluer en cours de l'exécution et plusieurs applications pouvant se partager l'infrastructure, nous avons pris le parti de n'avoir aucune connaissance des applications.
Si cette première approche tend à réduire la consommation de l'infrastructure en s'adaptant dynamiquement à son comportement, la consommation n'est pas proportionnelle à la charge. La consommation d'un noeud de calcul est définie comme étant :

$$P_h = P_{statique} + P_{dynamique}$$

où

$$P_{statique} = P_h^{min}$$

et

$$P_{dynamique} = (P_h^{max} - P_h^{min}) \times load_h$$

donnant donc

$$P_h = (P_h^{max} - P_h^{min}) \times load_h + P_h^{min}$$

avec P_h^{max} la consommation maximale de la machine h, P_h^{min} la consommation lorsque la machine est dans l'état idle (on parle de consommation statique) et $load_h$ sa charge.
Ainsi, même en supposant que les serveurs inutilisés sont éteints ; un serveur qui serait allumé mais ne fonctionnerait qu'à une faible charge aura une consommation qui pourra néanmoins être importante à cause de sa consommation statique.

Dans le deuxième axe, nous avons donc cherché à construire une infrastructure dont la consommation est proportionnelle à la charge. Cela, n'est pas possible avec les serveurs composant habituellement les centres de calcul : leur consommation dynamique dès qu'ils sont allumés est de base déjà élevée. De plus, si l'on veut obtenir des courbes de consommation proportionnelles, il est nécessaire d'introduire une grande diversité dans les consommations énergétiques des serveurs. Nous avons fait le choix d'étudier la composition de centre de calculs avec des architectures hétérogènes amenant une grande diversité de performance et de consommation. Cette approche nécessite de connaître le profil de charge de l'application et le profilage de sa consommation sur chaque type d'architecture.

La première partie du chapitre propose une méthodologie de réduction de la consommation énergétique des infrastructures de calcul à grande échelle prenant en compte tous les sous-systèmes composant un système HPC (ensemble de noeuds HPC) et sans connaissance à

priori sur les applications. La deuxième partie consiste à proposer la construction d'une infrastructure hétérogène permettant d'atteindre la proportionnalité énergétique.

La démarche scientifique utilisée a été similaire : expérimentations réelles sur la plateforme Grid'5000 pour la première approche et calibrations réelles puis simulation pour la deuxième approche. Les expérimentations en réel permettent d'avoir les valeurs réalistes des différentes architectures puis la simulation permet d'évaluer à plus grande échelle différents scénarios. La validation a été réalisée à l'aide de benchmarks standards de la suite NAS (NPB) et d'applications réelles.

2.2 Caractérisation d'application et leviers verts

2.2.1 Approches existantes

La modélisation de l'utilisation de puissance des applications a été investiguée dans le domaine mobile mais moins dans le domaine HPC. En effet, le problème est plus compliqué car une application peut s'exécuter sur plusieurs noeuds. Ces modèles sont basés sur l'observation du système pendant l'exécution de l'application à travers les compteurs de performance. Beaucoup d'approches logicielles se contentent de reconfigurer dynamiquement le processeur avec DVFS [KIS10], [FL05], [HK03], [RLdS+09], [KFL05]. On trouve des approches *off-line* dont l'intérêt est limité pour gérer la dynamicité du système et des approches *on-line*. Beaucoup d'approches *on-line* sont efficaces si le comportement des applications se répète et beaucoup mettent le focus sur l'application et non sur l'infrastructure complète qui peut être partagée entre plusieurs applications [ICM06], [LFL06], [SKK11].

2.2.2 Approche proposée

La méthodologie proposée a comme objectif d'exploiter tous les leviers possibles sur tous les composants d'une infrastructure HPC au niveau système et sans connaissance à priori sur les applications. L'objectif est de réduire la consommation d'énergie sans dégradation significative de la performance, ceci est relatif : une dégradation de 10% est souvent acceptable. Dans la suite de cette partie, lorsque l'on parlera de système HPC, on se réfèrera à un ensemble de noeuds de calcul et de stockage (sans prendre en compte l'infrastructure réseau). Lorsque l'on parlera de système, l'on se réfèrera à un noeud.

La méthodologie est basée sur trois étapes :

— Détection de phases : on appelle phase une période d'exécution où les métriques sont stables. Il s'agit ici de détecter des changements dans le comportement des applications.

— Caractérisation de phases détectées : il s'agit d'associer un label aux phases détectées afin ensuite de pouvoir appliquer des leviers verts appropriés.
— Identification de phases et reconfiguration (ou adaptation). L'objectif est d'identifier les phases récurrentes ; ceci est souvent utilisé en conjonction avec la prédiction.

Cette approche fait abstraction de l'application mais analyse le comportement du système en détectant des phases d'exécution, en analysant leurs caractéristiques afin d'adopter les leviers verts adaptés. Les travaux présentés ici s'intéressent aux infrastructures de HPC et proposent une extension pour le Cloud.

Détection de phases. La détection de phases demande souvent une connaissance de l'application, de l'architecture système ou la compilation de l'application avec des librairies d'instrumentation. Dans le cas d'une infrastructure HPC, c'est compliqué car l'infrastructure est souvent partagée entre des workloads différents en parallèle. Dans l'approche proposée, nous nous plaçons au niveau du système en détectant des phases en s'abstrayant des applications individuelles. Deux méthodologies sont proposées : une méthodologie hors-ligne appelée *power-based phase detection* et une méthodologie en ligne appelée *execution vector (EV) based phase detection.*

Power-based phase detection également appelée DNA [TCLP$^+$14b], [TCLP$^+$12b] décrit un noeud physique comme un graphe d'états où les états initiaux et finaux sont l'état idle. Chaque noeud du graphe représente le comportement du système pendant un intervalle de temps. L'idée est de détecter en mode hors-ligne les phases d'exécution en exploitant la consommation de puissance. Pour cela, on utilise les compteurs de performance et des métriques systèmes plus spécifiques comme "network bytes sent/received" et "disk read/write counts". La consommation de puissance est estimée par régression linéaire à partir des compteurs et métriques. Cette méthode a ses limites lorsque la consommation de puissance manque de précision. Pour résoudre le problème nous avons remplacé les valeurs de métriques par leur taux d'accès afin d'éviter des compensations entre valeurs. Ce concept est la base de la méthodologie suivante.

Execution vector (EV) based phase detection. Les compteurs de performance (Performance Monitoring Counters PMC) sont un ensemble de registres matériels conçus pour monitorer et enregistrer des événements bas niveau de performance se produisant à l'exécution au niveau du processeur. Ceci permet un monitoring très fin au niveau des processeurs multi-coeurs. Nous avons introduit le concept de vecteur d'exécution (EV) qui est un vecteur de taux d'accès aux compteurs de performance et "network bytes sent/received" et "disk read/write counts". Nous avons utilisé un critère de similarité ou de ressemblance entre EV en calculant leur distance de Manhattan (distance entre deux points dans un espace à n dimensions). Nous considérons qu'il y a un changement de comportement du système si les métriques changent de manière notable entre deux pas de temps consécutifs. Ainsi, dans notre approche, un changement de phase se produit quand la distance de Manhat-

tan entre deux EVs consécutifs est supérieure à un seuil. Ce seuil est fixé et est appelé *seuil de détection* ; c'est un pourcentage de la distance maximale entre EVs consécutifs. Cette distance maximale est réinitialisée à 0 à chaque changement de phase. C'est donc juste le pourcentage qui est fixé. L'algorithme 1 décrit la détection de phase proposée [TCLP+13].

Initialization : $max\_distance = 0$; $phase\_start = False$
`// threshold is a fixed percentage of the maximum existing distance max_distance`
while *True* **do**
  Compute EV_t : basically, sample a new execution vector
  Compute *dist* : the distance between EV_t and EV_{t-1}
  `//update the maximum existing distance max_distance`
  if $max\_distance \leq dist$ **then**
    $max\_distance \leftarrow dist$
  end
  if $dist \leq max\_distance * threshold$ ***and*** $phase\_start$ *is True* **then**
    Start a new phase
    $phase\_start = False$
  end
  else
    if $dist > max\_distance * threshold$ ***and*** $phase\_start$ *is False* **then**
      $phase\_start = True$
      `//reinitialize the maximum existing distance`
      $max\_distance = 0$
    end
  end
  $t \leftarrow t + 1$
end

Algorithm 1: EVPDA (EV-based Phase Detection Algorithm) : un algorithme en ligne pour la détection de changement de phase.

Pour évaluer cette méthode nous avons exécuté une série de benchmarks dont nous connaissions le comportement (donc les phases qui devaient être détectées) et nous avons mesuré trois métriques :

- la sensibilité : le pourcentage de changement de phases détectées par rapport au nombre de changements réels.
- les faux positifs : nombre de changements repérés qui ne correspondent à aucun changement réel.
- le temps de détection moyen : temps moyen que met l'algorithme pour repérer un changement de phase.

Les résultats de [TCLP+13] ont montré une sensibilité de 100%, un nombre de faux positifs

faibles correspondant aux phases "idle" et un temps moyen de détection entre 0,15 secondes et 0,5 secondes. La détection est en effet très rapide car elle consiste en la lecture de compteurs et le calcul de distance entre vecteurs dans un espace à n dimensions (n étant le nombre de compteurs et métriques utilisés).

Pour des phases de durée importante, les informations à stocker peuvent devenir trop couteuses. Nous avons donc défini un résumé d'une phase contenant notamment un vecteur de référence de la phase : le vecteur le plus proche du groupe de EVs de la phase. C'est ce résumé qui sera gardé en mémoire permettant ensuite de comparer les phases détectées et identifier les phases récurrentes.

Nous avons étudié l'influence du seuil de détection. Tout dépend de la granularité souhaitée dans la détection de phases mais il n'y a pas de relation linéaire entre le seuil de détection et le nombre de phases détectées comme a pu le montrer l'expérience menée avec l'application réelle WRF-ARW (Advance Research Weather Research and Forecasting) [SKD+05] au cours de laquelle nous avons fait varier la valeur du seuil (table 2.1) [TCLP+13].

TABLE 2.1 – Variation du nombre de phases détectées pour différentes valeurs de seuil.

Seuil de détection (%)	1	5	10	15	20	30	35	40	50
Nombre de phases détectées	1	2	13	12	27	52	1	1	1

Caractérisation de phases. L'objectif de cette étape est de regrouper les phases dans des classes : une classe correspondra à une utilisation similaire des ressources (processeur, disque, mémoire, réseau). Nous associerons ensuite à chaque classe des leviers verts. Les classes que nous avons considérées et leurs descriptions sont présentées dans la table 2.2.

L'objectif de la caractérisation est d'attribuer un label aux phases. Les labels *compute-intensive*, *memory-intensive* et *mixed* sont exclusifs.

Nous avons proposé différentes méthodes de caractérisation :

(1). Méthode *LLCRIR : Last Level Cache References per instruction* : la classification est basée sur la sensibilité au cache c'est à dire sur les statistiques de leur usage du cache (dernier niveau d'utilisation du cache). En exécutant des benchmarks que nous connaissions, nous avons défini des ordres de grandeur de leur LLCRIR par catégorie (la table 2.3 récapitule les valeurs) [TCLS13].

Nous avons vérifié que la classification ne changeait pas avec la taille des problèmes ni avec la fréquence du processeur. Les expérimentations ont montré que la méthode *LLCRIR* est utile pour déterminer si un workload est *memory-intensive*, *compute-intensive* ou *mixed* mais

TABLE 2.2 – Catégories de workloads HPC et leurs descriptions.

Catégorie de Workload	Description
Compute intensive	s'applique à toute application nécessitant beaucoup de calcul ; la performance est souvent limité par la vitesse du processeur.
memory intensive	s'applique à toute application demandant plus de mémoire que l'on ne peut en trouver sur les ordinateurs standards.
mixed	s'applique à toute application ayant les caractéristiques de memory intensive et compute intensive.
communication intensive/ network receive	s'applique à toute application nécessitant la réception de volumes de données importants par transferts réseaux.
communication intensive/ network transmit	s'applique à toute application nécessitant l'envoi de volumes de données importants par transferts réseaux.
IO intensive	s'applique à toute application nécessitant la lecture ou l'écriture de volumes de données important leur performance dépend de la vitesse des périphériques.

TABLE 2.3 – Ordre de grandeur des références LLC et labels associés.

Label du Workload	Ordre de grandeur de LLCRIR
Compute intensive	$\leq 10^{-4}$
memory bound	$\geq 10^{-2}$
mixed (à la fois memory et compute intensive)	10^{-3}

TABLE 2.4 – Règles d'association des labels aux phases en fonction des compteurs sélectionnés par ACP.

Compteurs sélectionnés par ACP pour la caractérisation de phase	**Label associé**
cache_ref & cache_misses & branch_misses or branch_ins	compute-intensive
no IO related sensor	communication IO intensive
branch_misses & hardware_ins or branch_ins	mixed
hardware_ins & cache_ref or cache_misses	memory-intensive

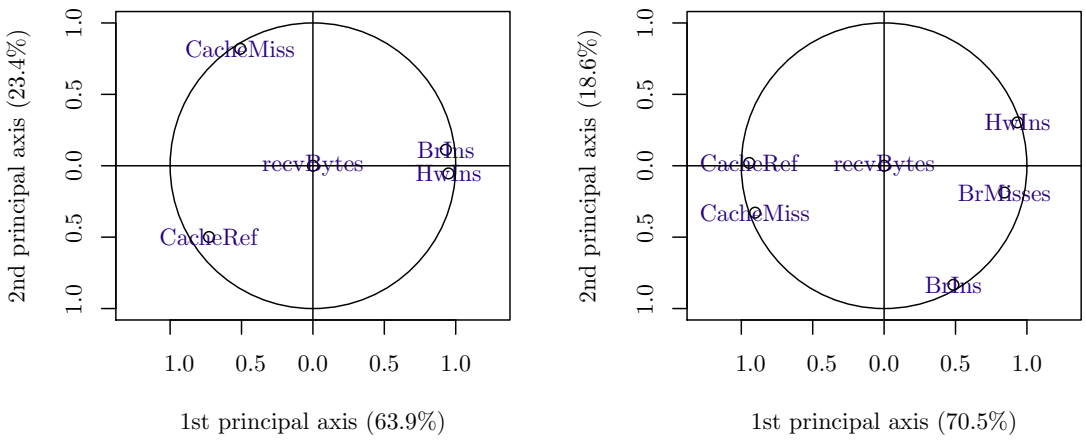
cela ne permet pas de savoir si le workload est *IO-intensive* ou *communication intensive*. La deuxième limite de cette méthode est qu'elle ne détecte pas les périodes idle.

(2). Utilisation de méthodes de statistiques ACP (Analyse en Composantes Principales ; en anglais PCA Principal Component Analysis).

ACP identifie des corrélations entre les données : cela permet d'identifier les directions principales de changement de données. Nous avons proposé deux méthodes basées sur l'ACP :

- Sensor based workload characterization : l'objectif est de trouver le comportement dominant d'une phase en interprétant les données collectées pendant la phase. Il s'agit d'appliquer la méthode ACP aux EVs, nous sélectionnons les 5 compteurs qui contribuent le moins au composant principal d'ACP : cela permet de savoir ce que le système ne fait pas. En fonction des 5 compteurs qui contribuent le moins au comportement de la phase on en déduit des labels. Les règles d'affectation des labels sont résumées table 2.4 [TCLP+12a], [DTCG+13].
  Malheureusement, cette méthode a les mêmes limites que la méthode *LLCRIR*.
- ACP based phase characterization. L'objectif est de découvrir des motifs partagés entre workload de la même catégorie. Nous avons appliqué la méthode ACP aux EVs collectés pendant l'exécution d'une suite de référence de benchmarks de chaque catégorie pour trouver comment les variables sont corrélées avec les composants principaux (PCs) de ACP [LVP+15]. La figure 2.1 montre la corrélation des compteurs avec le premier et deuxième axe principal d'ACP pour deux charges mélangées : l'application FT et BT de la suite de benchmarks parallèles NAS [BBB+91]. La figure 2.1 révèle que le premier composant principal (le 1er axe principal) oppose les compteurs liés au processeur (hardware instruction, branch instructions, and branch misses) à ceux liés à la mémoire (cache references and cache misses). Donc, nous pouvons supposer que pour des charges de travail mixtes, les compteurs liés au processeur sont opposés à ceux liés à la mémoire.

La caractérisation est fondamentale pour garantir une amélioration efficace de la perfor-



(a) Fourier Transform (FT). (b) Block Tri-diagonal solver (BT)

FIGURE 2.1 – Corrélation des compteurs.

mance et de l'énergie.

Nous avons proposé deux algorithmes de caractérisation basés sur les méthodes présentées précédemment. *Algorithme basé sur la règle de la majorité* applique les trois schémas de caractérisation et utilise la majorité pour déterminer le label. Lorsque les trois donnent des résultats différents nous utilisons le résultat de la caractérisation la plus fiable :

- LLCRIR est fiable lorsqu'elle indique qu'une phase est *compute-intensive*
- ACP-based est fiable lorsqu'elle indique qu'une phase est *memory-intesive*, *IO*, *communication-intensive* ou *mixed*.

Le deuxième algorithme proposé consiste à appliquer LLCRIR après ACP lorsque ACP échoue à trouver un label. Ces méthodologies sont rapides sans impact sur la performance et fonctionnent bien : elles permettent d'associer le bon label aux différentes phases et sont rapides (de l'ordre de la seconde) : le temps nécessaire est négligeable [TCLS14].

Identification de phases et leviers verts. Ceci est la troisième étape de la méthodologie proposée. Pour exploiter la détection de phases et la caractérisation il est essentiel d'identifier les phases récurrentes mais cela suppose que les phases identifiées soient terminées, ce qui n'est pas très utile pour un système en-ligne. La littérature suggère de prédire la prochaine phase mais ce n'est pas trivial. Nous proposons deux alternatives à la prédiction : *reconnaissance de phase partielle* et *classification de vecteur d'exécution*. Dans

TABLE 2.5 – Labels des phases et leviers verts associés.

Label de phase	Décision de reconfiguration possible
compute-intensive	switch off des barrettes mémoire inutilisées ; mettre les disques en mode sleep ; augmenter la fréquence du processeur ; utiliser le levier Low-power-idle pour les cartes réseaux.
memory-intensive	diminuer la fréquence du processeur ; diminuer la vitesse des disques ; ou les mettre en mode sleep ; allumer les barrettes mémoire.
mixed	allumer les barrettes mémoire ; augmenter la fréquence du processeur ; mettre les disques en mode sleep ; utiliser le levier Low-power-idle pour les cartes réseaux.
communication intensive	éteindre les barrettes mémoire ; diminuer la fréquence du processeur ; allumer les disques.
IO-intensive	allumer les barettes mémoire ; diminuer la fréquence du processeur.

les deux cas, lorsque l'identification de phase est terminée, nous utilisons la table 2.5 afin d'appliquer le levier vert approprié.

La reconnaissance de phase partielle consiste à identifier une phase en cours : on compare l'EV actuel avec les vecteurs de référence des phases mémorisées. Lorsque l'on a identifié une phase (utilisation de la distance de Manhattan), on arrête de comparer jusqu'à ce qu'on détecte une nouvelle phase [DTCG+13], [TCLP+14b].

La classification de vecteur d'exécution consiste à comparer chaque EV avec les vecteurs de référence des phases connues et ce pendant toute la durée de la phase.

Nous avons également proposé une identification de phase hors-ligne que nous appliquons systématiquement à toute nouvelle phase détectée pour éviter des caractérisations de phase inutiles. Cette identification hors-ligne est plus simple dans le sens où l'exécution de la phase et de toutes les phases sont terminées. Elle est basée sur le calcul de la distance de Manhattan entre vecteurs de référence des phases. L'objectif est d'enrichir notre base de connaissances de phases connues et caractérisées pour ensuite pouvoir comparer les nouvelles phases détectées avec le contenu de la base.

Différents leviers verts ont été expérimentés :

(1). Sélection de plate-forme. Nous supposons connaître une structure DNA de référence sur une plateforme donnée pour une application donnée. En comparant sur une portion d'exécution le DNA d'une application sur une plateforme avec celui sur la plateforme de référence on en déduit une estimation de la consommation d'énergie et une pertinence [TCLP+14b]. Une des limites de cette approche est que cela suppose que l'application est

la seule à s'exécuter.

(2). Memory-size scaling. Cela consiste à adapter la taille de la mémoire aux demandes du workload sur un principe similaire au processeur. Pour cela, nous avons expérimentalement mesuré les temps d'exécution de différents benchmarks avec différentes contraintes de RAM. Nous en avons déduit des tailles optimales de RAM (c'est à dire la taille de la mémoire à partir de laquelle le temps d'exécution restait constant). Nous en avons déduit qu'il était possible de réduire jusqu'à 10% la consommation énergétique. Ceci est un grand potentiel d'autant plus que les concepteurs de système HPC ne savent pas à l'avance combien de RAM les applications des utilisateurs utiliseront. Ce levier n'est actuellement pas disponible mais nous pouvons espérer que les futurs OS le permettront (quelques patchs pour Linux sont en cours d'évaluation).
Des efforts sont également en cours de la part des constructeurs pour diminuer la consommation de la RAM. La mémoire la plus commune utilisée dans les serveurs est la DRAM (Dynamic random-access memory). Les efforts actuels concernant DDR4 (même s'il n'est pas encore produit en grande quantité) permettront d'économiser de l'énergie : l'exécution se fait dans l'intervalle 1,05-1,2 V et opère à 3200 MHz. La plupart des noeuds des datacenters utilisent DRAM DDR3 (exécution dans l'intervalle 1,25-1,6 V et opère à 1600 MHz)[GDCR15].

(3). CPU cores switch on/off. Il s'agit d'allumer ou éteindre les coeurs des processeurs. Nous avons expérimentalement conclu que ce levier n'était pas pertinent pour des workload de type *compute-intensive*, *memory-intensive* et *mixed*. Par exemple, le benchmark LU a subi 58% de dégradation de performance lorsqu'il était exécuté sur 7 coeurs au lieu de 8.
Dans la génération précédente de processeurs multi-coeurs, on ne pouvait pas gérer les coeurs indépendamment (changement de C-state ou de P-state).

(4) DVFS : c'est un levier vert bien supporté qui consiste à modifier la fréquence du processeur. Sous Linux, il existe 5 modes de gouverneurs différents : trois modes sont statiques c'est à dire que la fréquence n'est pas modifiée au cours du temps ; deux modes sont dynamiques et permettent au gouverneur de changer la fréquence au cours du temps. Les cinq modes de gouverneurs sous Linux sont [gou] :

- PowerSave : utilisation de la fréquence la plus basse du processeur (mode statique)
- Performance : utilisation de la fréquence la plus haute du processeur (mode statique)
- Userspace : utilisation de la fréquence choisie par l'utilisateur parmi les fréquences possibles pour le processeur (mode statique)
- Conservative : mode dynamique qui fonctionne avec deux seuils (un seuil bas et un haut). La charge CPU est régulièrement comparée aux seuils ; lorsqu'elle est inférieure au seuil bas, la fréquence est diminuée à la fréquence la plus proche et inversement lorsque la charge est supérieure au seuil haut, elle est augmentée.
- OnDemand : mode dynamique fonctionnant de manière similaire à Conservative à la différence qu'il a un seul seuil.

(5) Network interconnection speed scaling : est un levier peu supporté par quelques NICs mais qui peut quand même être utilisé. Au niveau réseau, les routeurs ne sont pas *energy-aware*, c'est à dire qu'ils n'adaptent pas leur consommation à l'utilisation. Néanmoins, au niveau du réseau, il y a deux techniques pour réduire la consommation des cartes réseaux : l'*Adaptive Link Rate* c'est à dire la réduction dynamique de bande passante du lien en fonction du trafic instantané et le *Low-power-idle* qui consiste à transmettre le trafic à vitesse maximale et se mettre en mode "sleep" pendant les périodes idle. Beaucoup d'énergie est consommée pour maintenir allumée l'infrastructure réseau même lorsque celui-ci est sous-utilisé. On pourrait donc exploiter les états pour économiser de l'énergie. Néanmois, le changement d'état peut prendre 10 microsecondes, pour les applications de calcul dont la performance dépend du réseau cela peut engendrer des dégradations de performance.

(6) Disk spin-down : il s'agit d'exploiter les modes opératoires des disques ; le mode sleep permet de réduire la consommation d'énergie. Des efforts sont également en cours au niveau des disques avec le mécanisme DRPM (dynamic revolution per minute) ou en améliorant la consommation de la nouvelle génération de disque SSD. Cependant, les disques SDD ont encore des capacités limitées et ne sont pas utilisables dans les datacenters pouvant stocker des terabytes ou petabytes de données.

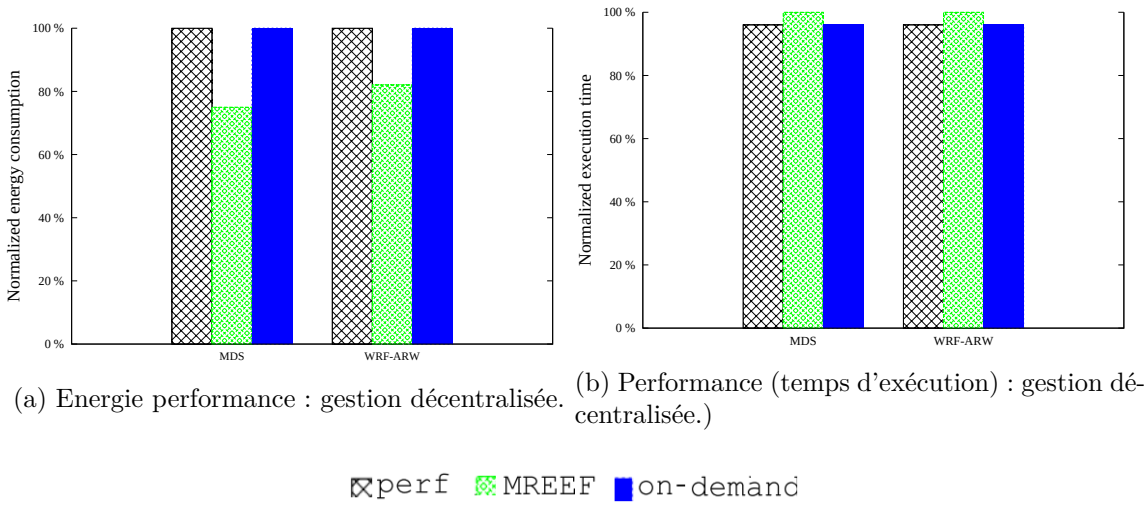
La méthodologie proposée a été implémentée dans le framework MREEF (Multi-Resource Energy Efficient Framework) [TCLP+14a]. La méthodologie agit sur tous les sous-systèmes de la plateforme HPC : le processeur, la mémoire, le réseau ...

L'évaluation s'est faite en comparant MREEF et les gouverneurs *on-demand* et *performance* disponibles sous Linux sur la plate-forme Grid'5000 [BCC+06].

MREEF était paramétré pour utiliser la détection de phase basée sur les vecteurs d'exécution, la caractérisation sensor-based et la reconnaissance de phase partielle et le levier DVFS (avec fréquence high pour les phases *cpu-intensive*, medium pour les phase *memory-intensive* et low pour les phases *IO et communication intensive*). Le seuil de reconnaissance a été fixé à 10%. Dans une évaluation centralisée, sur 15 noeuds (total de 60 coeurs) et avec en utilisant les benchmarks NAS, nous avons constaté que MREEF consomme 15% moins d'énergie en moyenne que les gouverneurs performance et on-demand pour une performance similaire (3% de dégradation).

Dans une évaluation décentralisée où chaque noeud est responsable de son profilage et de ses reconfigurations, sur 25 noeuds avec les applications MDS (Molecular Dynamics Simulation) [BHK+04] et WRF-ARW[SKD+05] nous obtenons en utilisant le levier processeur 19% de gain d'énergie pour 4% de dégradation de performance ; et en utilisant les leviers processeurs, disques et réseaux nous obtenons (figure 2.3) 24% de gain d'énergie pour une performance identique. Les résultats sont une moyenne de 20 exécutions de chaque workload sous chaque configuration système et sont normalisés avec "on-demand".

Cette étude a montré l'importance de considérer le système complet sans connaissance à



(a) Energie performance : gestion décentralisée. (b) Performance (temps d'exécution) : gestion décentralisée.)

FIGURE 2.3 – Détection de phase, reconnaissance partielle ; gestion décentralisée de MREEF.

priori des application s'exécutant et des possibilités de gain d'énergie sans dégradation de performance notable. Deux leviers de plus haut niveau n'ont pas été exploités : il s'agit de l'allocation de ressources dynamique qui sera présenté dans le chapitre 2 et le levier migration qui sera étudié dans la section suivante ainsi que dans le chapitre 2. Nous notons des perspectives intéressantes concernant les actions sur la mémoire lorsque les leviers seront opérationnels. Si cette approche globale permet d'économiser de l'énergie en actionnant des leviers verts appropriés, elle ne permet pas de diminuer le coût statique des serveurs. Elle permet de gagner de l'énergie pendant l'exécution des applications mais se focalise sur la partie dynamique de l'exécution. Il reste à étudier comment atteindre l'efficacité énergétique. Nous allons dans la section suivante présenter une approche visant à atteindre une consommation énergétique des serveurs proportionnelle à la charge en utilisant des infrastructures hétérogènes.

2.3 Energie proportionnelle

2.3.1 Approches existantes

Lorsque les serveurs sont allumés mais non utilisés dans un datacenter on gâche les ressources informatiques : en effet, l'électricité est consommée mais n'est pas transformée en puissance de calcul. Barroso et Holzle [BH07] ont montré que les serveurs sont rarement utilisés à leur puissance maximale. La charge moyenne des machines d'un datacenter de

Google est entre 10% et 50% or cela correspond à des charges où les serveurs sont le moins efficaces comme l'illustre la figure 2.4.

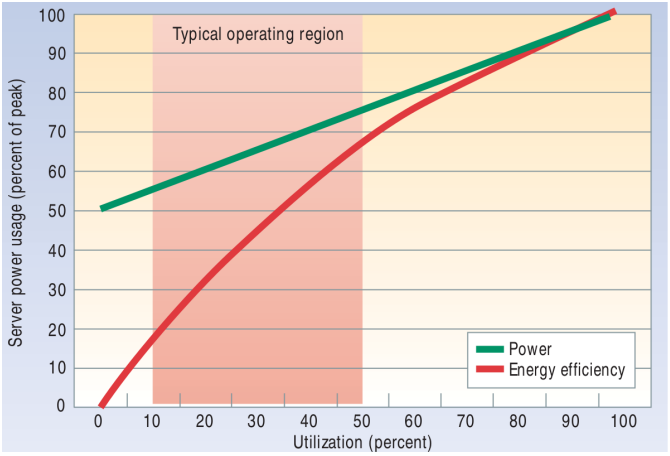


FIGURE 2.4 – Consommation des serveurs et efficacité énergétique pour des charges de 0 à 100% (Figure extraite de [BH07])

La problématique est donc la suivante : comme réduire les coûts statiques et obtenir une consommation proportionnelle à la charge ?

[VAG10] et [RPE11] ont proposé des métriques pour évaluer la proportionnalité d'une architecture. On constate que les serveurs sont globalement de plus en plus proportionnels mais comme une architecture purement proportionnelle n'existe pas encore une piste consiste à utiliser plusieurs architectures de performances différentes et de consommation différentes.

On retrouve cette idée d'hétérogénéité dans les processeurs multi-coeurs hétérogènes comme dans l'implémentation ARM avec big.LITTLE [big13] (même si ce sont les mêmes architectures ARM et les mêmes jeux d'instruction). La technologie big.LITTLE consiste à utiliser deux coeurs : un A7 lent à faible consommation énergétique et un A15 plus rapide mais ayant également une consommation plus importante. En migrant les applications entre les deux coeurs, le système d'exploitation optimise la consommation. Les processeurs hétérogènes constituent une approche innovante mais ils sont pour le moment dédiés aux dispositifs mobiles : tablette, smartphones dans l'objectif d'économiser leur batterie.

Dans la liste du Green500 [Grea], on peut constater que le haut de la liste est dominé par les supercalculateurs hétérogènes comme ceux couplant CPU et GPGPU. Le gros inconvénient d'un GPGPU est qu'il n'est pas possible aujourd'hui de l'éteindre lorsqu'il est inutilisé.

L'approche que nous proposons a comme objectif de proposer un système de placement dynamique des applications sur une infrastructure hétérogène composée de machines ARM et x86 avec une gestion intelligente d'allumage et d'extinction des ressources physiques. Les premiers aspects de ce concept ont été mené dans [Da 13] et ont été étendus aux infrastructures grande échelle en élargissant la gamme de processeurs dans [VDCL+14]. [Da 13] montre les avantages potentiels d'avoir un ensemble de ressources hétérogènes composé de Raspberry Pi, Intel Atom et Intel i7, hébergeant des serveurs Web sans état. La courbe de consommation d'énergie se rapproche de la proportionnalité et les gains d'énergie sont importants, particulièrement pour de faibles charges. Notre objectif principal est de poursuivre cette proposition, et d'étudier comment il est possible de l'étendre à une plus grande gamme d'applications. De plus nous voulons améliorer les résultats de [Da 13] en ajoutant la gestion de ressources avec les surcoûts d'allumage et d'extinction.

2.3.2 Approche proposée

Deux leviers verts sont utilisés dans cette approche où l'on construit dynamiquement l'infrastructure la plus proportionnelle possible : la migration et l'allumage/extinction. Dans notre approche, nous considérons une infrastructure composée de noeuds d'architecture différents ; dans [VDCL+14] nous en considérons trois auxquels nous associons des labels *Little*, *Medium* et *Big* mais ce nombre n'est pas une limite, l'approche peut être étendue. Nous imaginons avoir plusieurs noeuds de chaque type, auxquels le placeur peut avoir accès. L'objectif étant d'exécuter les applications sur l'architecture ou combinaison d'architectures la plus adaptée (c'est à dire celle qui consomme le moins pour les performances demandées). Les machines non utilisées seront mises en mode veille. L'application doit pouvoir être migrée si la charge varie.

Combiner des processeurs hétérogènes est une approche innovante car pour le moment ces architectures sont dédiées aux terminaux mobiles.

De nos jours, les datacenters sont surtout composés de serveurs avec des processeurs x86. Depuis les années 2000, presque tous ces processeurs, construits par Intel et AMD, ont un adressage mémoire de 64 bits. Ils ont une bonne performance par rapport à leur prix et sont les plus répandus. Cependant, leur inconvénient principal est leur consommation de puissance élevée lorsqu'ils sont inutilisés ou sous-utilisés. Nous allons donc chercher à exploiter des processeurs très basse consommation pour voir si nous pouvons neutraliser ces coûts statiques. Il apparaît que les processeurs ARM offrent le meilleur compromis entre la performance et la consommation énergétique. De fait, les processeurs ARM ont été historiquement conçus pour des systèmes embarqués donc la faible consommation de puissance était la principale contrainte. Maintenant ils sont de plus en plus conçus pour des dispositifs mobiles comme des smartphones et des tablettes, ainsi ils deviennent plus puissants. De plus, certains d'entre eux ont récemment inclus des extensions de virtualisation.

Ce dernier point a renforcé notre idée d'utiliser ces processeurs dans les datacenters.

Solutions de virtualisation pour architectures x86 et ARM. Notre première préoccupation a été d'étudier les solutions de virtualisation existantes pour vérifier la compatibilité avec des architectures ARM et x86 et savoir si l'on pouvait utiliser la migration à chaud *live migration* ou des mécanismes de point de contrôle/reprise *checkpoint/restart*. La virtualisation consiste à faire fonctionner plusieurs systèmes d'exploitation sur une machine physique. La *live-migration* consiste à déplacer l'exécution d'une machine virtuelle d'une machine vers une autre de manière transparente. Nous avons établi les spécifications nécessaires comme les systèmes d'exploitation, les versions de noyau, afin de voir quelle solution est la moindre restrictive. Notre objectif est de choisir une technologie sur ces critères, qui seront une bonne base pour développer une migration fonctionnant entre des architectures hétérogènes.

Nous avons considéré deux catégories principales : les machines virtuelles et les conteneurs d'application. Nous nous sommes concentrés sur des solutions open source, c'est pourquoi nous avons choisi KVM et l'hyperviseur Xen pour l'approche de machine virtuelle et LXC et OpenVZ pour les conteneurs.

TABLE 2.6 – Comparaison des possibilités de virtualisation ou conteneurs

	Virtual machines		Linux containers	
	Xen	KVM	LXC	OpenVZ
On x86	yes	yes	since 2.6.29	patched kernel
On ARM	since 3.7	since 3.9	since 2.6.29	patched kernel
Live migration	yes	yes	not yet (CRIU project)	yes, but not on ARM
Guest OS	any	any	only Linux based	only Linux based

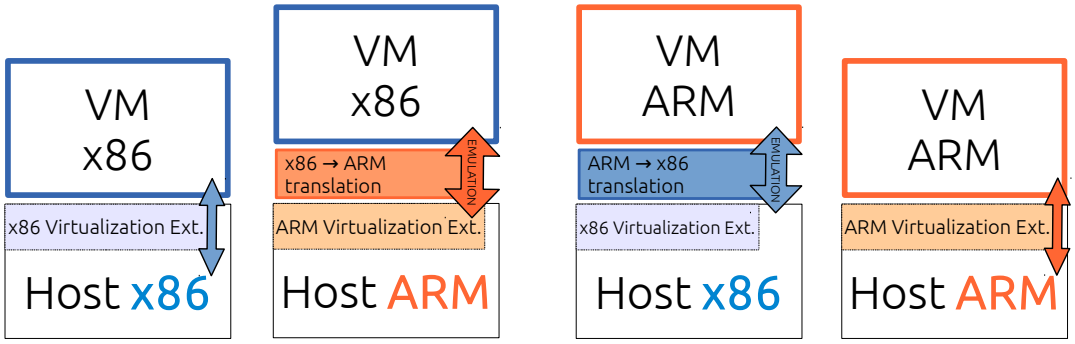
Bien que les conteneurs d'application semblent être une technologie prometteuse avec un processus de virtualisation très léger avec peu de surcouts, cela implique beaucoup de contraintes. Les conteneurs Linux fonctionnent uniquement avec des OS Linux pour lesquels l'OS hôte et invité partagent la même version d'OS et de noyau. De plus, le checkpointing pour des conteneurs est encore en cours de développement tandis que la *live migration* est bien aboutie pour des hyperviseurs comme Xen ou KVM. OpenVZ a une solution de *live migration* mais elle n'est fonctionnelle que pour des machines x86. En ce qui concerne LXC, le projet CRIU (Checkpoint/Restore In Userspace) [CRIrg] prévoit la mise en oeuvre de la *live migration*. Cette comparaison nous a conduit à choisir la solution de machine virtuelle, approche la plus commune dans les datacenters et approche imposant le moins de

restrictions. Les deux propositions KVM et Xen sont tout à fait équivalentes, nous avons choisi KVM.

Quand la machine virtuelle et la machine physique partagent la même architecture nous utilisons les extensions de virtualisation. Au contraire, si les deux architectures sont différentes, nous devons utiliser l'émulation. L'émulation est un concept qui permet d'exécuter des programmes compilés pour une architecture différente de l'hôte. Il consiste dans une abstraction du matériel permettant l'exécution du programme grâce à la traduction dynamique des instructions binaires.

Pour cela, nous avons choisi l'émulateur QEMU (qui est fortement lié à KVM). QEMU détecte si la machine virtuelle et l'hôte ont la même architecture, dans ce cas l'émulation n'est pas nécessaire et il utilise automatiquement les extensions virtualization. Par conséquent notre concept est basé sur l'hypothèse qu'il puisse être possible de migrer une machine virtuelle entre deux hôtes différents. Après la migration, le système devrait juste passer de l'émulation aux extensions de virtualisation, ou l'inverse, selon l'architecture des hôtes de destination et de la source. Les surcouts d'émulation ont été mesurés dans [VDCL+14] et des expérimentations sur la migration entre architectures hétérogènes ont été menées.

La figure 2.5 les différentes alternatives de fonctionnement.



(a) x86 VM (b) ARM VM

FIGURE 2.5 – Les deux alternatives : Emulation ou Extensions de Virtualisation.

Le système d'ordonnancement BML Nous proposons un système de placeur BML (pour Big, Medium et Little) dont les différentes briques sont exposées sur la figure 2.6.

Une première étape consiste à établir les profils des différents types de ressources : le profil de performance et de puissance, les surcoûts d'émulation, de migration et de on/off. Tous ces paramètres permettent ensuite d'établir un classement entre les différentes ressources

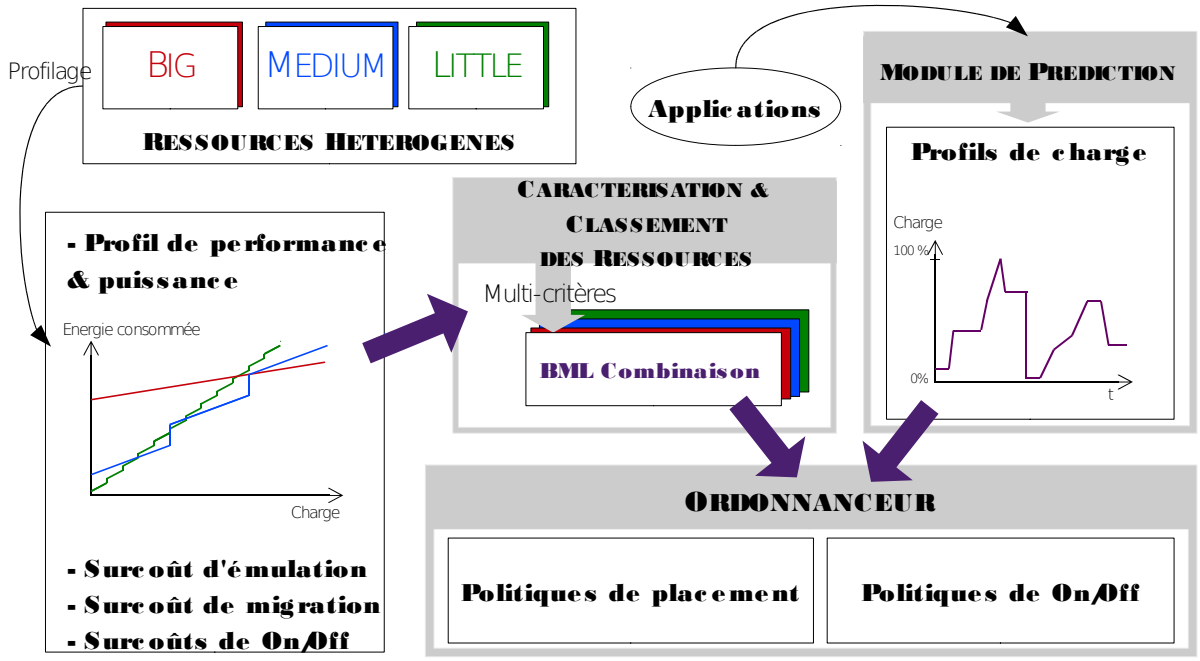


FIGURE 2.6 – Architecture du système d'ordonnancement BML

(affectation des *tags : Big, Medium, Little*). Ce classement est spécifique à l'application choisie comme cas d'étude. Les performances de chaque architecture ne sont pas les mêmes pour toutes les applications et le classement des architectures les unes par rapport aux autres peut différer en fonction de l'application choisie. Une fois que chaque architecture a été profilée individuellement, et classée par rapport aux autres, le but est de déterminer la combinaison d'architectures idéale. Le profil de charge au cours du temps des applications doit être connu afin d'adapter le placement. Nous supposons que ce profil est connu ou acquis par un module de prédiction *parfait*. C'est une hypothèse importante pour une approche "en-ligne" ; ainsi, soit il faudra disposer d'un bon système de prédiction soit l'approche sera utilisable dans les centres de calcul dont on connait bien l'utilisation.

Mise en oeuvre et validation Le tableau 2.7 récapitule les processeurs hétérogènes choisis : ARM Cortex-A15 pour le processeur de faible consommation (Samsung Chromebook sur lequel nous avons installé une distribution Linux). Pour les architectures x86, nous avons utilisés des noeuds de Grid'5000 [BCC+13] : un processeur Intel Xeon et un processeur AMD Opteron, localisés respectivement dans les clusters de Lyon et Rennes.

On observe une hétérogénéité dans les puissances consommées. En particulier la puissance consommée quand les machines sont sans activité : pour le serveur HP elle est 20 fois

TABLE 2.7 – Résumé des machines sélectionnées

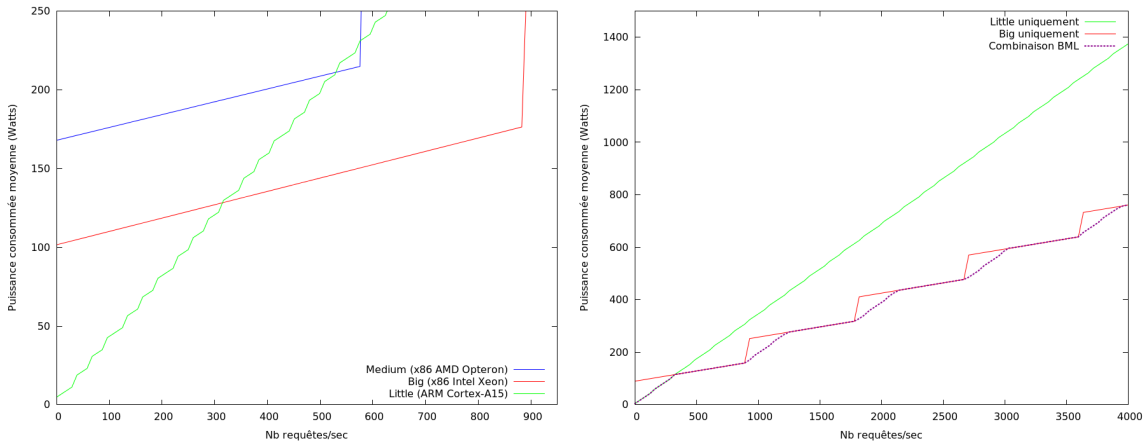
Nom	Samsung Chromebook	Dell PowerEdge R720	HP Proliant DL165 G7
Architecture	ARMv7 32 bits	x86 Intel 64 bits	x86 AMD 64 bits
CPU	2 x ARM Cortex-A15	2 x Intel Xeon E5-2630	2 x AMD Opteron 6164
Nb cœurs	2	12	24
Puissance	5 – 25 W	90 – 220 W	170 – 280 W
Année de sortie	2012	2012	2013

plus grande que celle du Chromebook, et 2 fois plus grande que celle du serveur Dell. La borne maximale de l'intervalle de puissance est la consommation quand tous les cœurs de la machine sont chargés au maximum.

Nous avons choisi un serveur web comme cas d'application ce qui offre une variabilité dans les requêtes nécessitant de faire correspondre la bonne architecture à l'évolution de la demande. Nous avons choisi le serveur web lighttpd. La page web interrogée est un script cgi écrit en python qui retourne aléatoirement une image parmi 5, de taille entre 1 et 3 ko. Pour générer la charge nous avons utilisé Siege qui est un outil de benchmarking web. Nous avons mesuré le profil de cette application sur chaque architecture afin de trouver quel est le nombre maximal de requêtes que peut traiter chacune architecture, ainsi que la puissance consommée pour faire ce traitement. Sur la figure 2.7a, nous avons représenté la consommation de chaque architecture en fonction du nombre de requêtes traitées. Une fois ces profils acquis, nous les interpolons : par exemple, le profil du processeur 'Little' basse consommation est interpolé en cumulant la performance et la consommation de plusieurs d'entre eux. Ainsi, nous pouvons constater que jusqu'à environ 340 requêtes par seconde, il est plus intéressant du point de vue énergie d'utiliser un ensemble de processeurs ARM plutôt qu'un seul processeur x86. Le deuxième constat est le fait que le profil du processeur AMD n'est pas intéressant dans ce cas-ci. En effet, il est moins performant que le processeur Intel mais sa consommation est la plus importante de toutes les architectures.

Avec ces profils nous pouvons savoir quelle est l'architecture, ou ensemble d'architectures la plus intéressante à choisir pour chaque niveau de performance. Nous appelons cet ensemble la combinaison BML. Le profil de la combinaison BML idéale est représenté en pointillés violets sur la figure 2.7b. Cette combinaison mélange uniquement les architectures les plus intéressantes des points de vue performance et énergie. Dans notre exemple seulement 2 types d'architectures ont été conservés : Intel Xeon ('Big') et ARM Cortex-A15 ('Little').

Le profil de charge dans l'application utilisée correspond au nombre de requêtes par seconde. Dans le cas d'une application de calcul, le profil de charge serait plutôt le nombre de MIPS utilisés au cours du temps. La démarche de recherche de la combinaison BML serait la même.



(a) Comparaison entre 3 architectures différentes (b) Création de la combinaison BML idéale

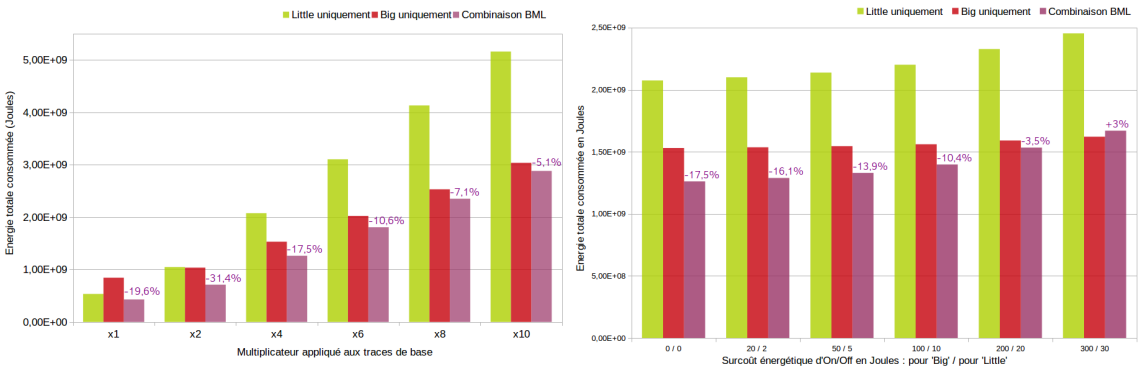
FIGURE 2.7 – Profils de puissance et de performance d'un serveur web lighttpd.

Pour valider à plus grande échelle, un simulateur en python a été développé. Ses entrées sont : les profils, le nombre de requêtes par seconde à traiter. Nous avons utilisé les traces de la coupe du monde de football de 1998 (les traces sont disponibles sur internet à l'adresse : `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`). En sortie, il fournit la consommation énergétique au cours du temps, la composition du datacenter et le nombre d'allumage/extinction.

Nous avons considéré dans ces expérimentations que nous avions un nombre illimité de machines de chaque type d'architecture. La configuration du datacenter est mise à jour chaque seconde en fonction du nombre de requêtes à traiter.

Nous avons appliqué un multiplicateur aux traces de base pour voir l'impact sur les gains relatifs de la combinaison BML par rapport aux configurations homogènes. Ces résultats sont présentés sur la figure 2.8a. Le pourcentage représente à chaque fois les gains de la combinaison BML par rapport à la configuration homogène la moins consommatrice. C'est uniquement pour les traces de base (multiplicateur x1) que la configuration 'Little' est la moins consommatrice, pour toutes les autres les pourcentages sont calculés par rapport à la configuration 'Big' uniquement. On voit sur la figure que plus le nombre absolu de requêtes est grand et moins les gains de la combinaison BML sont importants. Dans ces cas là, l'apport de l'architecture 'Little' n'est pas remarquable car l'architecture 'Big' sera majoritairement utilisée car elle est plus efficace pour traiter des taux de requêtes élevés. Par exemple, au cours du scénario où les traces ont un facteur multiplicateur de 10, le pic de charge génère environ 40000 requêtes par seconde conduisant un choix BML composé uniquement de 'Big' (43 machines). En revanche, dans le cas du scénario des traces de base, le pic de charge génère 4000 requêtes par seconde conduisant un choix BML de 4 'Big' et 9

'Little'. Bien entendu ces valeurs dépendent des ratios de performance et de consommation d'énergie de chaque architecture.
Ceci nous laisse penser que notre approche sera plus efficace dans des petits datacenters. En effet, dans les gros datacenters, la charge serait globalement trop élevée et conduirait à des combinaisons BML où les 'Big' seraient majoritaires. Dans tous les cas, il y aura toujours un seuil de charge à partir duquel les gains diminueront ; notre approche sera bénéfique lorsque la charge permettra d'avoir des combinaisons BML exploitant les différents labels.



(a) Différentes versions des traces de la Coupe du monde 98

(b) Différents surcoûts d'On/Off pour les nœuds 'Big' et 'Little' en Joules (traces x4)

FIGURE 2.8 – Energie totale consommée, et gains de la combinaison BML par rapport aux solutions homogènes pour différents scénarios.

Nous avons également intégré les surcoûts d'allumage/extinction des machines à notre solution.

Sur la figure 2.8b, nous montrons l'impact des surcoûts énergétiques d'On/Off. Nous avons choisi le scénario où les traces sont multipliées par 4. Nous avons testé différentes valeurs de surcoût pour voir les limitations de notre proposition. Par simplicité, nous avons considéré identiques les coûts d'allumage et d'extinction. Comme les architectures 'Big' et 'Little' ont dans notre cas un facteur 10 environ de différence en terme de consommation énergétique, nous avons supposé ce même facteur pour les coûts d'On/Off. Par exemple sur la figure, le cas '100/10' signifie qu'un surcoût de 100 Joules a été considéré pour l'allumage/extinction d'un nœud 'Big', et 10 Joules pour un 'Little'. Nous avons simulé pour les 3 configurations et calculer le pourcentage de gains de la combinaison BML. La configuration homogène 'Big' est toujours la moins consommatrice dans ce cas. La solution 'Little' est composée d'un nombre plus important de machines et de ce fait beaucoup d'allumages et d'extinctions sont nécessaires. Pour le cas où les surcoûts d'On/Off sont les plus grands, la combinaison BML consomme plus (3%) que la solution 'Big'.

Notre approche combinant des architectures hétérogènes pour constituer un datacenter

est innovante [VDCL+15]. Les expérimentations menées montre l'intérêt mais aussi qu'il est important de prendre en compte avec attention les surcoûts d'On/Off, et que notre solution doit être adaptée en fonction. Ici les décisions d'allumage et d'extinction sont prises à chaque seconde ce qui implique un nombre important de changement d'état des machines. Il est donc nécessaire d'explorer des systèmes de décisions moins fréquents et peut être ainsi plus efficaces.

L'approche peut être généralisée à plus de trois tags pour prendre en compte l'hétérogénéité des machines : on pourra ordonner les machines par profil ou éventuellement les regrouper en différentes classes de label dont les profils énergie/performance sont proches. Chaque classe étant triée par rapport aux autres et étant alors composée de plusieurs machines par profil ordonné de performance et puissance.

Bien entendu, restera à investiguer des configurations de datacenter avec des nombres limités de machines de chaque type. Il sera nécessaire à terme de considérer les impacts sur la qualité de service mais aussi les impacts de la prédiction sur la solution. La prédiction de charge est un domaine de recherche en soi et nous n'avons pas la prétention de développer notre propre module de prédiction. Plusieurs techniques peuvent être utilisées pour faire des prédictions basées sur des modèles différents comme le *Grey Forecasting* [JJFHHCLD14], ou ARIMA (Autoregressive Integrated Moving Average) [CMRB14]. Nous avons l'intention de tester des méthodes différentes avec des niveaux différents d'exactitude de prédiction dans notre simulateur pour voir les impacts sur les résultats. Nous pouvons aussi envisager d'utiliser l'approche présentée dans la section 2.2 pour repérer des phases dans le comportement des applications s'exécutant sur la plate-forme afin de choisir les bonnes architectures. Notre approche sera d'autant plus intéressante si elle minimise les surcoûts de mise en oeuvre notamment ceux relatifs à la migration à chaud. On trouve dans la littérature différentes façons de réduire le temps de migration. La gestion de la mémoire est un aspect important qui peut être amélioré, par exemple en utilisant des noeuds de stockage (Storage Area Network) [AHTH14], [OTS+11], ou en améliorant les pré-copies [HLS+14].

Par ailleurs, ce travail en cours, se poursuit en prenant en compte les caractéristiques des applications : sont elles parallélisables ? sont elles reconfigurables en un nombre fixé ou variable d'instances ? En effet, les applications sans état pourront être migrables plus facilement et le levier d'allumage/extinction pourra être utilisé plus facilement. Pour les applications avec état, la taille de l'état à migrer sera importante à prendre en compte dans le calcul des surcouts des leviers. L'approche sera plus intéressante pour les applications dont l'élasticité pourra varier dynamiquement. L'approche consistera à trouver la combinaison BML optimale pour les applications et à mettre en oeuvre une solution BML réelle la plus proche tout en prenant en compte les surcouts induits. Ainsi, par exemple le meilleur compromis choisi consistera peut être en une configuration BML intermédiaire. Tout ceci mettra en oeuvre des algorithmes que l'on pourrait comparer à des algorithmes

de placement sur lesquels nous nous attarderons au chapitre suivant. La reconfiguration de l'élasticité de l'application pour correspondre à la combinaison BML choisie pourra se faire grâce à des approches décrites dans le chapitre 4.

2.4 Projets et contextes applicatifs

Ces travaux ont utilisé la plateforme Grid'5000 pour de nombreuses expérimentations.
Le projet national Grid'5000 a permis de construire une plateforme expérimentale de recherche constituée d'une grille informatique de grande taille [CCD+05]. Des laboratoires de recherche et des universités françaises partagent des clusters (environ 1000 noeuds répartis sur 10 sites) : Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay (Paris), Rennes, Reims, Sophia-Antipolis et Toulouse. Les sites sont interconnectés par de la fibre optique dédiée (fibre dite "noire") à 10Gbps, ce backbone optique étant géré par Renater [GUI94]. L'intergiciel OAR (développé à l'IMAG de Grenoble) est utilisé pour réserver tout ou partie de cette grille [CCG+05].
Ces travaux ont été réalisés dans le cadre du projet INRIA HEMERA [Hem] de 2010 à 2014. Ce projet regroupe différentes équipes de recherche autour de différents challenges et groupes de travail. L'objectif était de démontrer l'efficacité des techniques scientifiques pour l'utilisation de plate formes "grande échelle" distribuées en effectuant plusieurs expériences de dimensionnement sur la plate-forme Grid'5000. L'objectif était également de participer à l'animation de la communauté scientifique autour de Grid'5000 et à l'agrandissement de la communauté en aidant les nouveaux venus à se servir de l'infrastructure.
Ces travaux ont été présentés dans le cadre de l'action COST IC804 *Energy efficiency in large scale distributed systems* (2009-2013). Cette action Européenne avait pour but de fédérer la communauté européenne sur la thématique de l'efficacité énergétique à grande échelle.

2.5 Bilan

2.5.1 Contributions et discussions

L'approche 2.2 est très intéressante, elle permet de considérer le système dans son ensemble sans connaissance à priori des applications. Plusieurs leviers verts ont été présentés et sont exploités dans ces travaux. Alors que la majorité des travaux de la littérature proposent des approches dédiées à des applications et centrées uniquement sur le processeur, nous avons proposé une méthodologie au niveau système de réduction de la consommation énergétique dans les systèmes distribués à grande échelle sans dégradation significative de la performance. Tous les composants du système sont pris en compte : processeur, mémoire, disque

et réseau. La méthodologie est basée sur une détection en ligne de phases d'exécution : le principe consiste à suivre le comportement du système HPC à travers la variation de l'utilisation des ressources (les compteurs de performance). Trois modèles de caractérisation de phases sont proposés : deux sont basés sur la méthode ACP (Analyse en composante principale) ; l'autre est basé sur les références au cache. En combinant ces modèles, différents algorithmes sont proposés (algorithme de la majorité ou ACP puis LLCRIR). Pour finir, nous avons proposé trois techniques d'identification de phases récurrentes : reconnaissance de phase partielle, classification d'EV (elle consiste à prédire l'EV de l'instant suivant) et identification hors ligne.

L'ensemble de la méthodologie a été implémentée dans le framework MREEF. L'approche a été validée avec différentes applications : des benchmarks standards de la suite NAS (NPB) et des applications réelles (WRF-ARW et MDS). Globalement les applications memory-intensive permettent plus de gains d'énergie que celles qui sont compute-intensive ou mixte. Nous avons expérimenté l'approche avec plusieurs applications HPC mais aussi Cloud [TCLS14] Dans le cas du Cloud, on notera toutefois que l'on atteint les limites de la caractérisation des applications : la catégorie memory-intensive étant très souvent choisie, cela permet moins de différenciation entre les workloads et génère donc un peu moins de gains que lorsque les usages de l'infrastructure sont diversifiés. Nous avons également étudié la performance sur une plateforme partagée par plusieurs applications, nous obtenons toujours de bons résultats [TCLP$^+$14b].

C'est une approche holistique intéressante ; toutefois en utilisant des leviers tels que le placement / la migration d'application nous pouvons aussi améliorer l'efficacité énergétique. C'est ce que nous verrons au chapitre suivant.

Ces travaux ont différentes perspectives : tout d'abord, la détection de phases pourrait utiliser des informations de température (vitesse des ventilateurs, température du serveur ...) et la reconfiguration déclenchée pourrait avoir comme objectif de faire baisser la température. En effet, nous verrons dans le chapitre suivant qu'une partie de l'énergie consommée sert à refroidir les serveurs.
La méthodologie proposée pourrait être améliorée grâce à des méthodes d'apprentissage. Un apprentissage hors-ligne ne serait pas intéressant. Afin d'éviter la perte de dynamicité ou de réactivité qu'un apprentissage en-ligne pourrait engendrer, procéder par fenêtre temporelle permettrait de borner l'apprentissage. L'apprentissage pourrait intervenir soit au niveau de la catégorisation de phase afin d'affiner les catégories prises en compte soit au niveau des leviers utilisables pour chaque catégorie.
Le framework MREEF pourrait être intégré dans un gestionnaire de cloud comme OpenNebula ou Openstack, ce qui permettrait d'activer les leviers adéquats dynamiquement au cours de l'exécution des machines virtuelles et ainsi économiser de l'énergie en s'adaptant à l'utilisation de la plateforme.
Nous avons évalué l'influence du seuil de reconnaissance sur la consommation et la perfor-

mance [TCLP+14b]. Il semblerait intéressant d'introduire un mécanisme de détection de seuil automatique. En surveillant l'impact des leviers verts mis en oeuvre ou la consommation d'énergie, on pourrait modifier dynamiquement le seuil. Nous reviendrons sur ce point au chapitre 5.

Cette approche est intéressante car elle permet de réduire dynamiquement la consommation du système en activant les leviers nécessaires : tous visent indépendamment à avoir une consommation proportionnelle à l'utilisation des différents composants. Cela améliore l'efficacité énergétique de l'infrastructure mais ne permet pas d'atteindre la proportionnalité car elle ne réduit pas le coût statique des serveurs. La deuxième approche le permet en construisant une infrastructure proportionnelle à base de serveurs hétérogènes. L'intérêt est immédiat, cela permet de réduire la consommation d'énergie statique c'est à dire lorsque les machines sont sous-utilisées. Nous utilisons pour cela le levier migration. La solution proposée repose sur des décisions de placement sur l'architecture adéquate. Pour cela, plusieurs conditions sont nécessaires : il est nécessaire de connaître le profil de charge de l'application et le profil de performance des architectures ; ensuite, des informations sur l'application doivent être connues : l'application est elle une application avec état (cela aura des conséquences sur la migration) ? quelle est l'élasticité de l'application : a t-elle un nombre fixé d'instances ? peut on la paralléliser ? Enfin, la décision de choix de composition de l'infrastructure implique une reconfiguration de l'application mais aussi des migrations ; ceci implique l'évaluation des coûts de transitions d'une configuration à une autre ainsi que l'évaluation du coût de la configuration cible afin d'évaluer le gain relatif potentiel. Pour ces derniers points, des solutions de l'autonomic computing que nous détailleront au chapitre 4 pourraient permettre de formaliser le style architectural de l'application et les reconfigurations possibles associées.

2.5.2 Encadrement et diffusion scientifique

Ces travaux ont été menés dans le cadre de deux thèses en collaboration avec l'équipe Inria Avalon du laboratoire LIP à Lyon et plus précisément avec Laurent Lefèvre.

- Ghislain Landry Tsafack : thèse commencée en 2011 et co-encadrée avec Jean-Marc Pierson et Laurent Lefevre du LIP à Lyon. La thèse a été soutenue en décembre 2013.
- Violaine Villebonnet : thèse commencée en 2013, co-encadrée par Jean-Marc Pierson, Georges Da Costa et Laurent Lefevre du LIP à Lyon. Je participe à l'encadrement des travaux.

Plusieurs publications ont été réalisées :

- 1 chapitre de livre : Large-Scale Distributed Systems and Energy Efficiency : A holistic view [12]
- 4 journaux : Parallel Processing Letters (à paraître en 2015) [2], Concurrency and

Computation : Practice and Experience (CCPE, 2014) [4] ; Future Generation Computer Systems (FGCS, 2014) [8] ; International Journal of High Performance Computing Applications (2013) [9].

— 8 conférences internationales : Parallel, Distributed, and Network-Based Processing (PDP 2015) [17] ; IEEE International Conference on Sustainable Computing and Communications (SustainCom 2014) [18] ; IEEE International Conference on Green Computing and Communications (GreenCom 2013) [23] ; Energy Efficiency in Large Scale Distributed Systems conference (EE-LSDS 2013) [25] ; International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2012) [26] ; IEEE International Conference on Parallel and Distributed Systems (ICPADS 2012) [27] ; International Workshop on Energy-Efficient Data Centres (E2DC 2012) [28] ; International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering (2011) [30].

Chapitre 3

Placement avec contraintes : énergie et thermique

3.1 Problématiques

3.1.1 L'importance du placement pour diminuer la consommation d'énergie

Au sein des infrastructures les deux grandes sources de consommation de l'énergie sont l'alimentation des serveurs et le refroidissement. Les études ont montré que les datacenters d'aujourd'hui consomment presque 2% de l'énergie mondiale [Koo11] et entre 30% et la moitié sert au refroidissement des machines [Saw04], [TGV08]. La majorité de la chaleur à évacuer est produite par les équipements informatiques. Le coût d'alimentation des serveurs est lié au problème de sur-provisionnement de ressources et de non proportionnalité de la consommation.

Dans le chapitre précédent nous avons exploité des leviers de l'infrastructure pour augmenter l'efficacité énergétique et obtenir une consommation proportionnelle à l'utilisation des ressources.
La proportionnalité ne peut réellement être atteinte que si on éteint les machines non utilisées. Dans les centres de calcul, les ressources sont sur-provisionnées pour pouvoir répondre aux pics de charge. S'il est simple d'éteindre les machines non utilisées, une gestion efficace des ressources est nécessaire : l'algorithme de placement a un rôle important : c'est lui qui choisit où exécuter les tâches sur les ressources physiques. Un placement qui consisterait à équilibrer la charge sur l'infrastructure ne permettra pas d'économiser autant d'énergie qu'un placement qui regroupera les tâches sur un sous-ensemble de machines et qui permettra ainsi d'éteindre les autres. La gestion de la charge de travail sur les machines

physiques grâce au placement permet donc d'optimiser la consommation énergétique lors de l'exécution pour améliorer l'efficacité énergétique en réduisant la consommation d'énergie des serveurs. L'idée est donc de réduire la consommation d'énergie de l'infrastructure en réduisant le nombre de machines allumées. Cette approche appelée consolidation a été rendue possible grâce à la virtualisation et à la migration à chaud *live-migration*.

Par ailleurs, le coût du refroidissement est un facteur limitant pour l'expansion des infrastructures. La gestion de la chaleur est très importante dans les infrastructures à grande échelle. Globalement, il y a deux moyens d'explorer le coût du refroidissement : le premier moyen est d'améliorer les méthodes d'extraction de chaleur (avec de nouvelles technologies de refroidissement ou en optimisant le système de refroidissement) ; le deuxième moyen est de fournir des solutions systèmes visant à contrôler la génération de chaleur par un placement efficace. Plus la charge des processeurs augmente, plus ils vont consommer de l'énergie et produire de la chaleur et plus le système de refroidissement devra fonctionner (donc consommer) pour maintenir une température souhaitée dans le centre de calcul. La chaleur produite par le processeur est proportionnelle à la puissance consommée qui est elle-même fonction de la fréquence du processeur. Ainsi, un levier utilisable pour réduire la consommation énergétique provenant du refroidissement serait le DVFS. Néanmoins, ce levier a un impact sur la performance. Une gestion intelligente des ressources : de la charge de travail des serveurs à travers le placement permettra d'éviter des points chauds et de diminuer la consommation d'énergie. S'il y a moins de points chauds, il y a moins de chaleur à évacuer et le système de refroidissement consomme moins.

3.1.2 Placement, réallocation et problème d'optimisation

Dans ce chapitre nous étudions l'efficacité énergétique au niveau des algorithmes de placement. Une politique de placement est caractérisée par ses objectifs d'optimisation.
On trouve dans la littérature deux grandes catégories d'algorithmes de placement : le placement statique et le placement dynamique. On parle d'ordonnancement lorsque l'exécution des tâches est planifiée dans le temps. Dans le cas du placement statique, la décision de placement est prise hors ligne, avant l'exécution sans prendre en compte l'état des ressources. C'est bien adapté aux applications déterministes dont le comportement est totalement maîtrisé mais cela ne prend pas en compte l'état des ressources.
Dans le cas du placement dynamique, la décision tient compte des informations disponibles sur les ressources, l'algorithme peut s'adapter au cours de l'exécution à la charge des processeurs : c'est un placement en-ligne.
Un problème de placement est un problème d'optimisation combinatoire défini par un ensemble de contraintes et une fonction de coût à minimiser (ou éventuellement maximiser). Le choix du placement peut se faire en cherchant à minimiser différents critères : le nombre de tâches en attente, le temps de réponse du système, le temps pour finir tous les processus déjà placés, la durée de l'application (le *makespan*), le temps libre des processeurs, la

quantité de ressources utilisées (le *sum-flow*) et bien entendu l'énergie consommée... Les algorithmes multi-critères permettent de placer les processus en tenant compte de plusieurs directives : la sélection des machines s'effectue en essayant de satisfaire au mieux les critères.
La formalisation de la fonction objectif d'un problème de placement multi-critères peut se faire de différentes manières (détaillées section 3.2.2) : en optimisant les critères par ordre de priorité, en optimisant un critère sous la contrainte d'un autre critère (critère borné) ou en utilisant une combinaison linéaire des différents critères.
Un problème de placement est un problème d'optimisation, problème connu pour être NP-complet c'est à dire que l'on ne sait pas résoudre en temps polynomial. Il existe différents types d'algorithmes pour résoudre les problèmes NP-complet [Tou10] :

- les algorithmes exacts : ils fournissent des solutions exactes en un temps de calcul non borné. L'exploration arborescente et la programmation linéaire font parties de cette catégorie. Ces algorithmes sont envisageables sur des problèmes de petite taille.
- les algorithmes d'approximation garantie : ils fournissent des solutions approchées dont la qualité est bornée par rapport à l'optimal.
- les heuristiques et les métaheuristiques : ils calculent des solutions approchées avec des temps de calcul raisonnables sans garantir la qualité des solutions.

Les métaheuristiques sont des schémas algorithmiques pouvant s'appliquer à différents problèmes d'optimisation. Elles proposent des stratégies de recherche génériques permettant d'éviter les minima locaux. On peut citer le recuit simulé, la recherche tabou, les algorithmes évolutionnaires (dont les algorithmes génétiques font parties), les colonies de fourmis... La plupart des métaheuristiques nécessitent de régler des paramètres de résolution dont dépendent la qualité de la solution et ayant un impact sur la rapidité du calcul de la solution. Bien souvent, les métaheuristiques permettent d'obtenir de meilleures solutions que les heuristiques mais avec des ordres de grandeur de résolution plus importants.
Une heuristique simple et rapide que l'on trouve dans la littérature : l'algorithme glouton *greedy algorithm* construit une solution étape par étape en choisissant de manière itérative un optimum local et ce sans jamais remettre en cause les décisions prises à l'itération précédente.

Les algorithmes de placement traditionnels se concentrent principalement sur l'allocation initiale des machines virtuelles, néanmoins, il est intéressant voire nécessaire d'étudier la gestion des machines virtuelles pendant leur exécution. La solution d'un placement en-ligne a malheureusement une durée de vie très (trop) courte car les systèmes évoluent rapidement, l'utilisation des ressources est dynamique remettant ainsi en question les choix d'optimisation précédents. Par conséquent, il est nécessaire de ré-évaluer le placement réalisé soit périodiquement soit lorsque certains événements sont reçus dans l'objectif d'optimiser l'utilisation des ressources notamment en utilisant la migration de machines virtuelles ; on parle alors de réallocation.

Le problème théorique sous jacent au problème de la consolidation est un problème de *vector packing* : les tâches sont caractérisées par leur besoin en CPU et mémoire. Le problème de vector packing a été largement étudié dans la communauté de recherche opérationnelle. Pour le résoudre des heuristiques gloutonnes simples sont souvent utilisées comme first-fit decreasing (FFD) ou best-fit decreasing (BFD). Les deux méthodes fonctionnent suivant un principe similaire : les tâches sont triées par ordre décroissant de ressources demandées, puis l'algorithme les place dans l'ordre. Dans first-fit, la tâche est placée sur la première machine permettant de l'exécuter. Dans best-fit, la meilleure machine (selon différents critères) est choisie. Ces algorithmes ne sont pas optimaux, mais ils sont très rapides et permettent d'obtenir de très bons résultats en pratique. La qualité du placement obtenu dépend de l'intelligence du tri utilisé. Il est possible pour des problèmes de petite taille (nombre réduit de tâches et de machines physiques) de calculer l'optimal avec des méthodes de programmation linéaire : dans sa thèse Damien Borgetto [Bor13] a évalué le temps de résolution de l'optimal pour différentes tailles de problème. Le temps de résolution explose dès que l'on considère plus de 8 machines et 16 tâches du fait du grand de combinaisons possibles.

Dans notre cas, il s'agira de minimiser l'énergie totale consommée. Généralement, les objectifs des politiques économes en énergie peuvent être divisés en trois catégories : celles se focalisant sur l'énergie *energy aware*, celles prenant en compte la chaleur *thermal aware* et celles considérant l'environnement (impact écologique) *environment aware*.

Les politiques *energy aware* peuvent se concentrer sur la minimisation de l'énergie totale (qui peut inclure seulement l'équipement informatique ou l'infrastructure de centre de données entière), limitant l'utilisation de puissance réelle ou cherchant un compromis entre la performance et l'énergie consommée. Les politiques *thermal aware* prennent aussi en considération le transfert thermique qui affecte en fait la consommation d'énergie totale plutôt que l'utilisation d'énergie brute. Les objectifs peuvent consister à minimiser la chaleur dissipée, à réduire le déséquilibre de production de chaleur (pour éviter des points chauds) ou à considérer des métriques spécifiques relatives à la chaleur.

3.1.3 Notre proposition et nos hypothèses

Nous avons choisi d'étudier comment grâce au placement nous pouvions diminuer la consommation d'énergie totale du centre de calcul sans dégrader la performance : en plaçant les tâches avec des contraintes de chaleur et en mettant en oeuvre la consolidation nous agissons sur les deux sources principales de consommation d'énergie : celle provenant des serveurs et celle provenant du refroidissement.
Les problématiques sont donc les suivantes : quels algorithmes de placement peut-on proposer pour l'allocation initiale des machines virtuelles, pour la réallocation dynamique permettant la consolidation ?

La consolidation est bien étudiée dans la communauté, nous avons choisi de l'étudier en prenant en compte tous les surcoûts engendrés par la consolidation : cout de migration, surcoût d'allumage et d'extinction. En effet, une gestion intelligente des machines est nécessaire : des extinctions/allumage en boucle et trop rapprochés seraient contre-productif étant donné le temps et la consommation d'énergie lors de la mise en oeuvre de ce levier. Nous avons donc étudié différentes politiques de gestion des machines. Plutôt que de modéliser tous les coûts induits : durée d'activation du levier, consommation d'énergie, indisponibilité ... nous avons pris le parti d'avoir une politique de gestion des machines pro-active. Nous cherchons à anticiper les évolutions de charge pour éviter des surcouts d'allumage et d'extinction.

Les décisions sont prises en ayant très peu d'information sur les applications : uniquement l'expression de leurs besoins en ressources mais sans connaissance précise du type d'application ou des communications engendrées.

Nous proposons de résoudre le problème de réallocation par un algorithme de consolidation basé sur un algorithme glouton en appliquant un tri des machines intelligent puis en plaçant les tâches de manière à résoudre le déséquilibre en ressources.
Différentes formalisations du problème de placement dynamique sont proposées : la première consiste à minimiser le critère de performance (le makespan) sous la contrainte d'une température maximale des noeuds de calcul. La deuxième consiste à construire une fonction de coût composite novatrice inspirée de la logique floue.

Le point commun de nos travaux est la proposition d'heuristiques gloutonnes. Dans nos travaux, nous considérons des problèmes pour des systèmes grande échelle. Ainsi, les méthodes de résolution à l'aide de programmation linéaire ne sont pas appropriées. L'étude de la réallocation se situe dans le contexte du Cloud Computing tandis que l'étude de l'allocation dynamique se situe dans le contexte HPC. Ainsi, dans les travaux que nous présentons, dans un cas, plusieurs VMs peuvent s'exécuter sur la même machine physique tandis que dans l'autre il y a une seule tâche par machine. Toutefois, il faut préciser que les heuristiques proposées sont transposables à d'autres contextes, seules des contraintes supplémentaires seraient introduites ou relâchées dans la formalisation du problème.

Dans ce chapitre, les critères de performance que nous avons choisis sont le temps d'attente des tâches et le makespan. Tous les travaux présentés dans ce chapitre allouent les tâches sur les machines au fur et à mesure de leur arrivée dans le système. Le placeur n'a aucune connaissance des tâches futures.

Nous avons validé nos propositions en deux étapes : grâce à des expérimentations réelles à petite échelle puis à plus large échelle par simulation. Cette démarche a plusieurs avantages : elle permet d'obtenir des valeurs réalistes pour la migration et les allumages/extinctions et de se comparer à un algorithme de gestionnaire de cloud. La simulation permet ensuite de confirmer l'intérêt de l'approche tout en supprimant les effets de bords qui peuvent se

produire lorsque l'on a une plate forme de taille trop réduite. Les expériences concernant la prise en compte de la chaleur ont été réalisées uniquement par simulation car la propagation de chaleur sur une petite plate forme ne permet pas d'évaluer ces algorithmes. Par contre, la simulation reproduit des effets réels de propagation de chaleur. L'ensemble des travaux du chapitre évaluent les performances de nos heuristiques pour différentes charges de travail générées aléatoirement et comparées avec soit des algorithmes de la littérature : best fit, fastest, greenest, coolest soit des algorithmes réels : l'algorithme fourni par OpenNebula décrit section 3.3.4.

3.2 Approches existantes

3.2.1 Allocation et réallocation de machines virtuelles

De nombreux travaux étudient la gestion des centres de données et des nuages pour optimiser la consommation énergétique tout en respectant des compromis de QoS, notamment à travers l'allocation de machines virtuelles (VM) [ODdAL14], [MOC+14], [Gue14], [KMT14] et la reconfiguration de l'infrastructure.

[BAB12] présente les notions de provisionnement de VMs, d'allocation et réallocation de VMs sur des machines physiques. Les auteurs utilisent la ressource processeur afin de consolider dynamiquement les VMs, réduisant ainsi significativement la consommation d'énergie globale de l'infrastructure. De plus, les auteurs mettent en œuvre une politique de replacement de VM basée sur un double seuil (bas et haut), aussi bien qu'une politique de minimisation de migration, pour limiter le nombre de migrations provenant d'un hôte surchargé. Leur évaluation est basée sur deux métriques : la consommation d'énergie totale de l'infrastructure et le pourcentage de violation de SLA (service-level agreement). Les résultats montrent des économies d'énergie de 66% comparé à une approche sans migration, en gardant un taux bas de violations SLA.

[XSC13] propose un algorithme de consolidation basé sur une approche avec différents seuils pour représenter les "points chauds et froids". Ils réallouent les machines virtuelles pour réduire les points chauds et combinent cette approche avec de la prédiction de charge de machines virtuelles afin d'obtenir une meilleure gestion de ressources.

Yang et al. dans [JSPJJ12] proposent un algorithme de placement basé sur un algorithme de *bin packing* afin de fournir une heuristique (appelée SEP-pack) pour la consolidation. Ils prennent en compte les caractéristiques de l'application : application CPU intensive ou application data intensive et gèrent les VMs différemment en allouant d'abord les VMs data intensive puis en allouant le reste sur les ressources restantes.

Dans [GHZ13], les auteurs présentent deux algorithmes pour la réduction de consommation

d'énergie et la réduction de coût de migration dans le nuage. Ils calculent une solution exacte basée sur la programmation linéaire entière pour minimiser la consommation d'énergie par des VMs caractérisées par des contraintes de CPU, mémoire et disque. Leur approche permet d'assigner un budget d'énergie à chaque hôte, qui ne doit pas être dépassé. Ils comparent leur approche avec un algorithme *best-fit*.

Une autre approche de consolidation par programmation linéaire est proposée dans [TTS+12]. Les auteurs étudient la consolidation des VMs en modélisant la consommation d'énergie et le coût d'une reconfiguration par migration. Ils proposent deux algorithmes : un calculant l'optimal et une heuristique basée sur un algorithme glouton calculant une approximation plus rapide. Ils comparent les algorithmes en utilisant les traces du supercalculateur T2K-Tsukuba sur les critères de consommation d'énergie, la dégradation de performance et le temps de calcul. Les résultats montrent une réduction de la consommation énergétique entre 15% et 50% ; l'heuristique étant meilleure pour la consommation énergétique alors que l'optimal est meilleur pour le critère de dégradation de performance.

Si toutes ces approches sont très proches de la notre car elles utilisent la consolidation, des différences existent concernant les caractéristiques des machines virtuelles prises en compte mais la grande différence réside dans le fait que notre approche intègre tous les surcouts induits par la consolidation : surcout de migration, d'allumage et d'extinction de machines.

3.2.2 Placement multi-objectifs

Le placement avec plusieurs critères conflictuels a été beaucoup étudié dans la littérature. Nous présentons différentes méthodes de formalisation du problème :

(1). *Priorité simple.* Ceci est une approche à base de priorité simple visant à optimiser des objectifs multiples dans l'ordre. Assayad [AGH04] a présenté une fonction de compromis bi-critères pour mettre des priorités entre le makespan et la fiabilité pour l'ordonnancement d'applications temps réel. Pour minimiser l'émission de carbone et maximiser le bénéfice, Garg [GYAB11] a proposé des politiques en deux étapes pour placer les applications dans des datacenters hétérogènes en utilisant des priorités relatives des deux objectifs. Dans [DSH+13] les auteurs ont proposé des heuristiques pour optimiser la qualité de service pour des services interactifs avant de considérer la consommation d'énergie sur des processeurs multi-coeurs avec DVFS.

(2). *Le front de Pareto.* Cette approche est souvent utilisée en mode hors-ligne pour produire un ensemble de solutions non dominées. Durillo [DNP13] a appliqué cette technique au compromis entre le makespan et la consommation d'énergie pour des serveurs hétérogènes. Gao [GGQ+13] a utilisé la méta-heuristique basée sur les colonies de fourmis pour obtenir le front de pareto pour le gaspillage de ressource et la consommation énergétique

dans le placement de machines virtuelles. Des algorithmes évolutionnaires ont été employés dans [YKB07, GS12] pour obtenir un ensemble de solutions alternatives pour le placement de workloads scientifiques sur des environnements de grille. Dans [KMT13], les auteurs proposent une approche d'allocation de ressources pour cloud basée sur trois critères : énergie, émissions CO2 et profit.

(3). *Optimisation contrainte.* Cette approche optimise un objectif soumis à la contrainte de l'autre. Rizvandi [RTZL10] l'a appliqué pour minimiser la consommation d'énergie soumise au makespan pour un placement initial. "A mixed integer programming model" a été utilisé par Petrucci [PCL+11] pour réduire la consommation électrique de serveurs virtualisés soumis à des contraintes de QoS. Fard [FPBF12] a développé une stratégie double pour minimiser la distance Euclidienne entre les solutions produites et un ensemble de contraintes dans un problème d'optimisation à quatre objectifs.

(4). *Combinaison pondérée de critères.* Cette approche combine des objectifs multiples dans un seul. Lee et Zomaya [LZ11] ont utilisé DVFS pour résoudre le compromis entre le makespan et la consommation d'énergie en considérant une somme pondérée des deux objectifs. La même technique a été utilisée par les auteurs de [ALW10, SCH09] pour une minimisation en ligne d'un objectif combinant le temps de réponse des tâches et l'énergie. Une approche semblable a été prise par Sheikh et Ahmad [SA12], qui ont considéré un objectif supplémentaire de température maximale dans un système multi-coeurs, optimisant par conséquent trois objectifs en même temps.

Nous appliquerons dans la section 3.4 une nouvelle approche de résolution multi-critères en l'appliquant au critère thermique. La logique floue [XF10, ZCL+13] présente l'avantage de permettre l'expression de préférences contextuelles entre critères. L'approche que nous présentons en est inspirée et est innovante quand une légère priorité existe entre critères.

3.2.3 Prise en compte de la chaleur

La prise en compte de la chaleur dans les algorithmes de placement pour datacenters à grande échelle a été étudiée ces dernières années [SAWR12]. Wang [WvLD+09] a envisagé de réduire les températures maximales/moyennes en minimisant le temps de réponse des tâches. Ils ont proposé une heuristique de placement simple qui alloue les tâches "chaudes" pour "refroidir" des noeuds de calcul et ont évalué leur performance par la simulation. Dans [WKD12], des techniques de *backfilling* ont été utilisées pour planifier des applications parallèles. Cependant, aucune propagation de chaleur d'un noeud à ses voisins n'a été prise en compte dans leur modèle.

Moore [MCRS05] a pris en considération la recirculation de chaleur et proposé des algorithmes de placement dont *minHR*, pour réduire la recirculation de chaleur et le coût de

refroidissement d'un datacenter. Un outil de prédiction appelé *Weatherman* [MC06] a été utilisé pour prédire les aspects thermiques d'un datacenter en utilisant des techniques de *machine learning*. Ils ont montré que l'outil prédit précisément la distribution de chaleur du datacenter sans nécessiter la configuration thermique statique. Un algorithme de placement basé sur *Weatherman* réalise des performances semblables à *minHR*.

Tang [TGV08] a aussi étudié le problème de minimisation du coût de refroidissement dans les datacenters en considérant la recirculation de chaleur. Ils ont caractérisé le comportement thermique des datacenters par une matrice de distribution de chaleur [TMGC06]. Ils ont proposé des algorithmes de placement hors-ligne basés sur les algorithmes génétiques. Leurs solutions ont été évaluées en utilisant une matrice de distribution de chaleur d'un datacenter de petite échelle. La même matrice a été utilisée dans nos travaux pour évaluer nos heuristiques de placement.

Pakbaznia [PP09] minimise l'énergie totale d'un datacenter tant celle consommée par les serveurs de calcul que celle consommée par les unités de refroidissement. Ils ont montré que la consolidation pour éteindre des serveurs inoccupés couplée à un placement tenant compte de la redistribution de chaleur peut significativement réduire la consommation énergétique totale. Banerjee [BMVG11] a étudié le placement de tâches tenant compte de la chaleur en explorant le comportement dynamique de l'unité de refroidissement d'un datacenter (CRAC *Computer Room Air Conditioner*).

En comparaison des travaux précédents, qui se concentrent surtout sur le placement hors-ligne dans des datacenters homogènes, nous avons étudié le problème en ligne pour des datacenters hétérogènes. Nous explorons dans la partie 3.4 le compromis entre la performance et l'énergie en utilisant une approche souple inspirée de la logique floue avec des priorités. De plus, nous avons aussi étudié dans la partie 3.4.3 le placement statique de serveurs pour équilibrer la distribution de chaleur en présence d'une matrice non-uniforme. A notre connaissance, aucune approche antérieure n'a considéré ce problème pour des datacenters hétérogènes.

3.3 Placement efficace en énergie

Les algorithmes proposés dans cette section ont un objectif d'efficacité énergétique et intègrent différentes considérations :

- Consommation d'énergie : la consommation d'énergie doit être optimisée et doit inclure tous les surcouts pouvant être induits.
- Nombre de migrations : l'objectif est d'éviter un goulot d'étranglement sur le réseau par le dimensionnement et la limitation des migrations faites dans un certain intervalle de temps.

— Qualité de Service : les utilisateurs exigent une certaine quantité de ressources pour leur VMs qui doit être allouée pendant la durée de l'exécution.
— Temps de calcul : la solution doit être calculée rapidement afin de pouvoir être mise en oeuvre lorsqu'elle est encore pertinente.

L'algorithme de consolidation proposé est basé sur un algorithme de "Vector-packing". Nous avons étudié le comportement des migrations sur une plate-forme réelle pour paramétrer l'algorithme de consolidation. Nous mettons aussi en œuvre une stratégie de gestion des hôtes pour éviter des actions d'allumage et d'extinction inutiles. L'évaluation est faite sur la consommation d'énergie et la durée effective des VMs, pour différentes charges de l'infrastructure.

3.3.1 Le modèle

Nous considérons un centre de données composé de H machines ayant la possibilité d'être allumées ou éteintes automatiquement. Les VMs ont une durée d'exécution finie, au cours d'une expérimentation de durée T, J VMs seront soumises. L'algorithme de placement proposé est chargé de placer les VMs soumises périodiquement : à chaque appel périodique du placeur, celui-ci étudie le meilleur placement pour les VMs en attente et les place (il n'y a pas de placement dans le futur). Chaque VM exprime un besoin de ressources qui dans notre est cas est : CPU, mémoire. Le CPU est une ressource qualifiée de fluide (l'application fonctionne avec des performances dégradées si elle obtient tout ce qu'elle a demandé) alors que la RAM est une ressource rigide (si la quantité allouée est inférieure à ce qui était demandé cela posera des problèmes d'exécution). Nous avons choisi dans ces travaux de considérer les deux ressources comme des ressources rigides afin de garantir un niveau de QoS par rapport au besoin exprimé par les applications. Nous notons vm^{CPU} le nombre de CPU demandés par une VM et vm^{MEM} le nombre de MB de RAM demandés. Le placement des VMs sur les machines doit respecter les demandes de ressources. Nous notons $e_{vm,h} = 1$ si la VM vm est placée sur la machine h, sinon $e_{vm,h} = 0$.

Notons h^{CPU} la capacité processeur d'une machine et h^{MEM} la capacité mémoire, la charge est définie par l'équation :

$$r \in R = \{CPU, MEM\} : \quad load_h^r = \sum_{vm \in J} \frac{vm^r \times e_{vm,h}}{h^r}$$

avec

$$load_h^r <= 1$$

La durée d'une VM est définie par l'équation :

$$d_{vm} = t_{sched} + t_{boot} + t_{run}$$

Dans laquelle t_{sched} représente le temps mis par le placeur pour trouver la solution de placement, t_{boot} représente la durée de démarrage de la VM et t_{run} correspond au temps d'exécution de la VM.

3.3.2 Les algorithmes

Nous proposons deux algorithmes : un algorithme pour une allocation des VMs en attente basé sur une heuristique "best-fit" et un algorithme de consolidation présenté dans l'algorithme 2 qui correspond à une reconfiguration périodique des VMs en cours d'exécution afin d'optimiser les migrations à faire pour la réduction d'énergie sans devoir réallouer toutes les VMs. Si un placement n'est pas trouvé pour une VM en attente, elle devra attendre jusqu'à l'itération de planification suivante. Il pourrait donc y avoir un risque de famine pour les VMs, mais ce cas correspondait à un système mal dimensionné avec trop de VMs.

L'algorithme de réallocation a l'objectif d'apporter des économies d'énergie en utilisant la consolidation pour réduire le nombre de machines utilisées par les VMs quand l'état de l'infrastructure a changé lorsque des VMs se sont terminées libérant ainsi des ressources. L'algorithme proposé (algorithme 2) résout le problème de "vecteur packing" (à deux dimensions : CPU, RAM), il redistribue les machines virtuelles d'une ou plusieurs machines pour réduire le nombre de machines allumées.

L'approche utilisée est basée sur l'algorithme présenté dans [LKK99] et fonctionne en séparant les machines dans autant de listes que nous avons de dimensions pour l'allocation (en autant de listes que le nombre de ressources considérées pour l'allocation). Chaque liste contient les machines ayant le plus de disponibilité dans une ressource particulière. Dans notre cas, nous aurons deux listes, une pour les machines qui ont plus de processeur que de mémoire et une pour l'inverse. Les VMs sont allouées sur les machines conformément au tri de façon à réduire le déséquilibre de ressources. Par exemple, si une VM demande $vm^{CPU} > vm^{MEM}$ l'algorithme choisira de préférence une machine de la liste où $load_h^{MEM} > load_h^{CPU}$ réduisant ainsi le déséquilibre dans la consommation des ressources.

L'algorithme prend en compte un nombre de machines qu'il essaie de décharger. Ce nombre peut être choisi de différentes manières : à l'aide de seuils précisant la charge minimale acceptée sur une machine soit en calculant un nombre de machines à garder allumées. Nous avons choisi la deuxième possibilité : nous calculons donc le nombre de machines minimum à garder allumées afin de répondre aux demandes de VMs auquel nous rajoutons un nombre appelé "pivot" afin d'optimiser les allumages et extinction de machines. Ce nombre doit varier en fonction du nombre de machines physiques et du nombre de VMs. Il doit être choisi pour minimiser les opérations d'allumages et d'extinction afin d'optimiser

begin Consolidate VMs
  H = sort(H, load ascending);
  $llh = h \in H$ where min($load_h^r$);
  $\{H^{CPU}, H^{MEM}, Migrations\} = \emptyset$;
  for $h \in H, h \neq llh$ **do**
    if $load_h^{CPU} > load_h^{MEM}$ **then**
      add h to H^{CPU};
    else
      add h to H^{MEM};
    end
  end
  success=True;
  foreach $vm : (e_{vm,llh} = 1)$ **do**
    found = False ;
    if $vm^{CPU} > vm^{MEM}$ **then**
      found = Try to find a suitable host h in H^{MEM} first, then H^{CPU};
    else
      found = Try to find a suitable host in H^{CPU} first, then H^{MEM};
    end
    if $found$ **then**
      add (vm, h) to $Migrations$;
      change list of h if needed ;
    else
      success=False;
    end
  end
  if *success=True* **then**
    enforce $Migrations$;
  end
end

Algorithm 2: SOPVP : Algorithme de réallocation

les surcouts induits. De plus, il devra aussi dépendre du nombre de migrations simultanées qui peuvent être faites vers une machine sur l'infrastructure de réseau. Les migrations de VMs (live-migration) mettent une tension sur le réseau en transférant les pages mémoire. Le nombre de machines choisi pour la consolidation tient compte du nombre maximal de migrations concurrentes autorisées et doit donc être adapté à la taille de l'infrastructure. L'algorithme 2 présente une consolidation avec un nombre de machines à décharger égal à 1, ce qui peut se généraliser. Les migrations sont réalisées si et seulement si la machine peut être entièrement déchargée. C'est un choix d'optimisation. Le but est d'éviter les migrations qui pourraient être non pertinentes à long terme parce que d'autres décisions peuvent être meilleures dans la boucle de planification suivante.

Nous avons expérimenté dans la section 3.3.3, l'influence des surcouts.

3.3.3 Etude des surcoûts des leviers verts

Nous avons présenté dans la section précédente un algorithme de placement basé sur le principe de consolidation qui intègre deux leviers verts qui sont : allumage/extinction de machines et migration. Nous avons étudié l'influence des surcouts de ces leviers afin de les appliquer de manière optimale. Nous avons fait les expériences sur une plate-forme RECS [REC14] contenant un nuage déployé avec OpenNebula sur KVM, comprenant 6 machines Intel i7-3615QE (4 cœurs). Les valeurs de consommation électrique sont mesurées pour chaque processeur avec un Plogg watt-mètres. Les images des VMs sont dans un espace de stockage NFS partagé. Chaque machine a 4 CPU et 15.6 GB de RAM.

Les métriques

Dans cette section nous allons expliquer les métriques utilisées pour évaluer l'algorithme de consolidation proposé et mesurer les surcoûts des leviers verts. Nous définissons deux métriques : une pour évaluer la QoS et une pour mesurer la consommation d'énergie. Nous avons défini une contrainte sur l'allocation des tâches : une VM ne peut obtenir moins de ressources que ce qu'elle a demandé. Notre modèle ne connait pas l'application s'exécutant dans la machine virtuelle ; nous avons donc défini *la durée effective* d'une VM comme métrique de QoS :

$$d_{vm}^{eff} = \frac{t_{run}}{d_{vm}}$$

La durée effective est calculée comme étant la durée réelle à partir du moment où la VM s'exécute divisé par la durée totale, qui est la durée du boot de la VM, jusqu'à son effacement. Une durée effective de 90% signifie que la VM a passé 90% de son temps dans l'état running.

L'autre métrique que nous voulons évidemment pour notre modèle est la consommation d'énergie de l'infrastructure. Il y a deux types de consommation d'énergie que nous obtiendrons ou calculerons. Le premier est la consommation d'énergie réelle, calculé comme la somme de toutes les consommations énergétiques mesurées sur tous les hôtes.

$$E = \int_0^t P_t dt = \int_0^t \sum_{h \in H} P_h dt$$

Nous utiliserons aussi la consommation énergétique simulée. Etant donné que les surcoûts d'allumage et d'extinction ne peuvent être négligés [LO10], il peut être difficile de quantifier l'amélioration de consommation d'énergie qu'un algorithme de consolidation apportera. Nous calculons la consommation énergétique simulée en considérant que les machines s'allument et s'éteignent immédiatement et sans surcoût. Nous simulons les états des machines et calculons la consommation.

$$E^{sim} = \int_0^t P_t^{sim} dt = \int_0^t \sum_{h \in H} \sum_{vm \in J} e_{vm,h} \times P_t dt$$

Allumage et extinction

Dans cette section, nous étudions le surcout de l'actionnement du levier marche/arrêt des machines physiques. Nous avons évalué différentes stratégies de gestion des hôtes. Il y a plusieurs façons de manipuler l'état des machines, il s'agit de choisir quand allumer ou éteindre les machines pour économiser le maximum d'énergie en coordination avec l'algorithme de consolidation [LO10].

— FirstEmpty : cette première stratégie est la plus simple mais pas la plus efficace. Elle consiste à éteindre les machines dès qu'elles sont vides et à les rallumer dès que la charge globale de l'infrastructure est supérieure à un certain seuil (dans l'expérimentation que nous avons faite nous avons pris un seuil de 75%).

— Pivot : cette deuxième stratégie calcule d'abord le nombre estimé de machines nécessaires pour exécuter toutes les VMs et consiste à garder ce nombre de machines allumées plus un nombre supplémentaire appelé "pivot". Ceci évite une consolidation trop forte des VMs qui conduirait ensuite à des allumages et extinction en boucle qui seraient très couteux [BMDC+12]. Le nombre estimé de machines nécessaires pour supporter la charge induite par les VMs à exécuter est calculé avec la formule :

$$host\_number = \max_R \left(\frac{\sum_{vm \in J} vm^r}{h^r}\right)$$

Il s'agit du maximum de la somme des besoins ressources de chaque VM. Le nombre pivot est quand à lui choisi en fonction de la variabilité de la charge et de la taille de l'infrastructure.

— Variation : la dernière stratégie de gestion de machine est basée sur la variation de la charge globale au fil du temps. Cela garde l'historique de toutes les variations qui se sont produites et compte le nombre d'augmentations de charge et le nombre de diminutions. S'il y a plus de diminutions que d'augmentations, il essaiera d'éteindre une machine et vice-versa.

Nous avons comparé les différentes stratégies : leur effet sur la consommation d'énergie et sur la durée effective moyenne pour une charge moyenne donnée (50%) dans l'infrastructure en utilisant l'algorithme de consolidation présenté.

La stratégie Pivot s'est avérée correspondre au meilleur compromis dans l'objectif d'éviter des surcoûts d'allumage et d'extinction [BS14]. C'est donc cette politique de gestion des machines qui a ensuite été utilisée pour valider l'approche globale. En effet, il n'est pas pertinent de consolider de manière abusive les machines virtuelles sur les machines physiques de façon à garder une certaine disponibilité en cas de pic de charge. Tom Guerout [Gue14] définit le Dynamisme ou "Capacité latente" dans sa thèse pour représenter la capacité de calcul disponible sans allumer de nouvelles machines physiques.

Migration

L'objectif de l'étude est d'analyser le nombre de migrations concurrentes réalisables sur une infrastructure : c'est à dire le nombre de machines que l'algorithme de consolidation peut chercher à "décharger" en même temps. L'objectif est de ne pas dépasser un nombre de migrations simultanées qui engendreraient un surcout intolérable. C'est une étude dépendante de la plate-forme qui devrait servir à calibrer les algorithmes. Pour cela, nous avons instancié avec OpenNebula plusieurs VMs sur la plate-forme RECS et fait varier le nombre de VMs à migrer vers une machine aléatoire. Nous avons mesuré le temps de migration moyen des machines virtuelles migrant simultanément d'une machine vers une autre [BS14].

3.3.4 Les résultats

L'approche proposée a été validée de deux manières : à travers des expérimentations réelles et à travers la simulation à plus grande échelle.
Dans les deux cas, nous avons évalué :

— les gains de l'algorithme proposé à travers les deux métriques performance et énergie par rapport au placeur par défaut d'OpenNebula et par rapport aux algorithmes de la littérature.
— l'impact de la stratégie de gestion des machines.

Expérimentation réelle : comparaison avec OpenNebula

Dans cette section nous présentons une validation de l'algorithme proposé en le comparant avec le placeur par défaut d'OpenNebula appelé *mm_sched* et décrit en détails dans [ONE14].

L'algorithme *mm_sched* procède en trois étapes :

- Filtre les machines qui n'ont pas suffisamment de ressources
- Trie les machines conformément à la politique de tri choisie (voir ci-dessous).
- Les machines avec le plus fort rang sont choisies pour l'allocation.

L'ordonnanceur a plusieurs politiques : la première est *Packing* qui vise à minimiser le nombre de machines utilisées. Son opposé est *Stripping*, qui vise à maximiser les ressources disponibles pour les VMs sur les machines. L'heuristique *Load-Aware* essaie de maximiser les ressources disponibles pour les VMs en utilisant les machines les moins chargées.

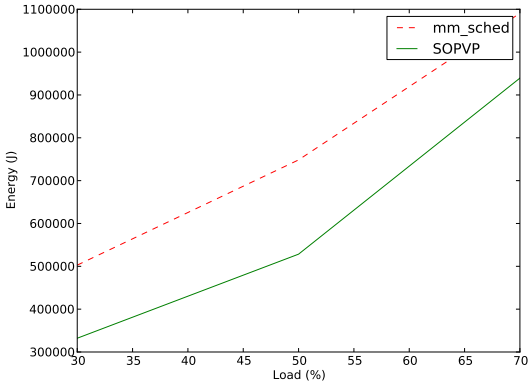
Chaque expérience est faite sur différents algorithmes : nous comparons l'algorithme proposé avec *mm_sched* d'OpenNebula avec l'heuristique stripping. Nous avons choisi de nous comparer avec l'heuristique stripping car c'est celle qui donnera la meilleure QoS aux VMs. Ceci dit, il aurait aussi été pertinent de se comparer avec la politique *packing* puisque c'est une heuristique de consolidation.

Pour chaque algorithme, nous mesurons différentes métriques : la consommation énergétique et la durée effective des VMs.

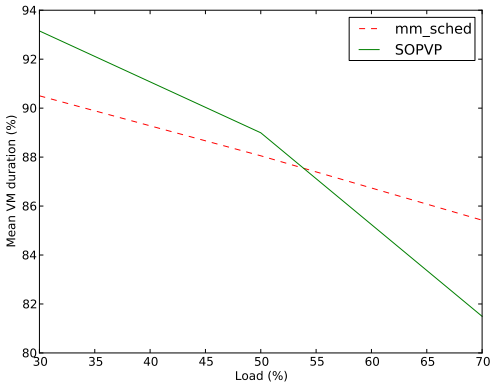
Nous avons mené deux expérimentations : dans la première expérimentation, nous avons exécuté les VMs et calculé une énergie consommée simulée P_t^{sim} introduite dans la section des métriques 3.3.3 ; c'est à dire que les machines ont un allumage/une extinction sans surcout. Cela nous a permis d'évaluer l'efficacité de l'algorithme indépendamment des surcoûts. Dans la deuxième, l'allumage et l'extinction des machines ont été implémentés permettant de vérifier si l'approche était toujours intéressante malgré les surcoûts. Ce qui nous permis d'évaluer l'efficacité réelle de l'approche complète : algorithme de consolidation et politique de gestion des machines.

Nous générons aléatoirement la charge sur l'infrastructure en créant des VMs qui sont "CPU-stressed" afin d'émuler une activité dans la VM. La génération des VMs suit une loi de Poisson, avec un taux d'arrivée λ adapté pour produire une charge moyenne souhaitée. Chaque VM durera entre 300 et 500 secondes, les durées suivent une distribution uniforme de moyenne 400 secondes. Chaque VM demande entre 0.1 et 2 CPU selon une distribution uniforme et demande entre 1024 MO et 6624 MO. Chaque VM est conçue pour représenter en moyenne 25 % des capacités d'une machine physique.

Chaque VM est soumise pour la durée de l'expérience ; les placeurs les alloueront/migreront selon une boucle de placement de 30s. Chaque algorithme de placement utilise le même



(a) Consommation énergétique pour différentes charges



(b) Durée des VMs pour différentes charges

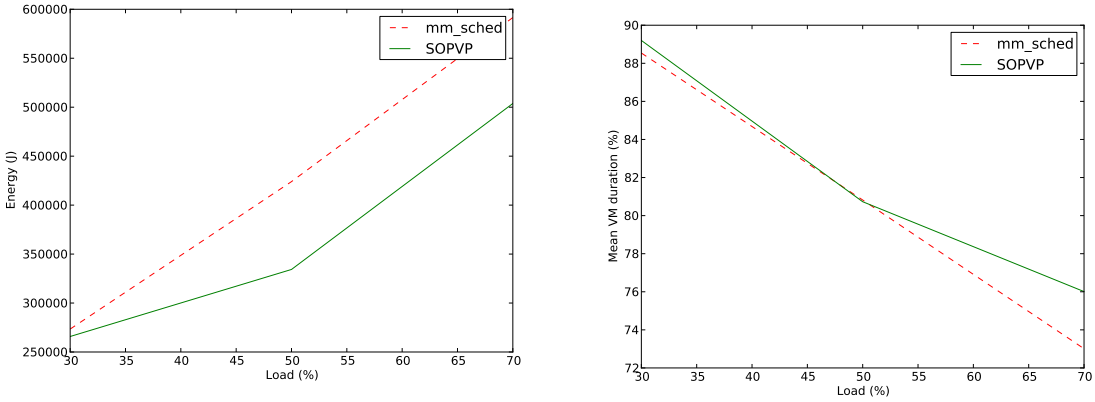
FIGURE 3.1 – Consommation d'énergie simulée et durée des VMs pour différentes charges.

"workload" généré. Chaque expérience dure 2h et a été reproduite 3 fois. Les expérimentations représentent 17 expériences différentes ce qui représente plus de 34h.

Energie simulée (sans surcoût d'allumage/extinction). La figure 3.2a présente la consommation d'énergie pour différentes charges. Etant donné la taille de notre plateforme expérimentale, l'algorithme de consolidation est paramétré pour décharger une machine. Nous pouvons constaté que la consommation augmente avec la charge. L'algorithme SOPVP est plus efficace de 36% pour de faible charge et de 16% pour de forte charge.

Cependant, ces résultats sont à confronter avec ceux de la figure 3.2b qui trace la durée effective des VMs. Comme attendu, avec une augmentation de la charge nous voyons une diminution dans la durée effective moyenne des VMs. Ceci est dû aux pics de charge qui rendent l'allocation initiale plus difficile à réaliser. Nous pouvons aussi voir que même si SOPVP est meilleur pour des charges basses, la différence entre les deux diminue quand la charge augmente, tendant même à une inversion pour des charges élevées. Notamment parce que SOPVP fait plus de requêtes à OpenNebula, ralentissant ainsi la décision dans l'attente des réponses de ONE.

Energie réelle (avec allumage et extinction). Nous avons implémenté l'allumage et l'extinction des machines de manière à mesurer la consommation d'énergie réelle afin de vérifier si les gains d'énergie constatés avec la mesure simulée se maintenaient. Nous avons utilisé la stratégie de gestion des machines "Pivot" avec un nombre pivot égal à 1.



(a) Consommation énergétique pour différentes charges

(b) Durée des VMs pour différentes charges

FIGURE 3.2 – Consommation d'énergie réelle et durée des VMs pour différentes charges.

La figure 3.2a représente la consommation d'énergie pour les charges de 30%, 50% et 70%. La consommation d'énergie est toujours meilleure avec l'algorithme SOPVP. Cependant pour une charge de 30% la différence de consommation d'énergie entre les deux algorithmes est moins importante que celle constatée avec l'énergie simulée. Ceci est du au fait que la charge moyenne est autour de 30% mais oscille entre 10% et 50%, ainsi étant donné que nous avons seulement 6 hôtes, il est difficile d'avoir moins de 2 hôtes utilisés. Ainsi les surcouts d'allumage et extinction auront plus d'impact. Pour une charge de 50%, nous économisons 21% d'énergie.

La figure 3.2b représente la durée effective moyenne des VMs pour différentes charges. Globalement, SOPVP permet de maintenir la même QoS que MM_SCHED, mais la QoS baisse lorsque la charge augmente. Lorsque la charge augmente, l'algorithme MM_SCHED a plus de difficultés que SOPVP à trouver une allocation laissant en attente les tâches pour lesquelles le placement a échoué.

Il faudrait noter ici que même si une durée effective moyenne autour 70-80% peut sembler mauvais, c'est principalement en raison du fait qu'avec une durée de VM autour de 400 secondes, une boucle de planification toutes les 30 secondes, 30 secondes pour allumer une machine et entre 30 secondes et 1 minute pour OpenNebula (temps d'ajout des machines au groupe de machines disponibles ...) nous pouvons avoir un impact important du temps d'attente dû à la plate-forme expérimentale, puisqu'il représente une proportion importante comparée à la durée moyenne. Un tel effet serait grandement atténué si nous avions pris des VMs avec des durées moyennes plus importantes.

Une variabilité non négligeable peut se produire sur la plate-forme. En effet, il y a de petites

variations dans les réponses d'OpenNebula ou même une variabilité dans le temps mis pour démarrer une VM ou une machine. Les difficultés liées à l'expérimentation réelle ne nous ont pas permis de reproduire plus d'expériences ni d'avoir des expériences plus longues. Ces expérimentations nous permettent néanmoins de vérifier en réel que notre approche de consolidation tenant compte des surcouts des leviers verts est intéressante. Nous l'avons expérimenté sur des infrastructures plus large-échelle à travers la simulation en utilisant le simulateur DCWorms. C'est que nous présentons dans la section suivante.

Simulation avec DCWoRMS

Nous avons repris notre expérimentation sur les RECs avec le simulateur DCWoRMS (Data Center Workload and Resource Management Simulator)[KOP+13]. DCWoRMS est un simulateur développé au laboratoire Poznan Supercomputing and Networking Center (PSNC) de Poznan en Pologne permettant de tester des politiques d'allocation et d'ordonnancement de ressources pour les plateformes HPC. Il permet la modélisation et la simulation d'infrastructures de calcul afin d'estimer la performance et différentes métriques de consommation d'énergie pour différents workload et différentes politiques d'ordonnancement.

C'est un simulateur développé initialement pour des plates-formes HPC, nous avons adapté certains mécanismes afin de pouvoir utiliser le simulateur.

Tout d'abord, la définition du "workload" a été modifié. Dans les expériences sur la plateforme RECS, nous créions des VM et les supprimions lorsque son "walltime" était atteint. Dans DCWoRMS, les tâches sont définies par une demande en CPU, mémoire, une date d'arrivée et de fin. Nous attendons donc toujours la fin d'une tâche. Nous ne pourrons donc plus utiliser la métrique de QoS définie précédemment ; désormais la métrique de QoS sera le temps d'attente c'est à dire : la durée entre une arrivée de tâche dans le système et le début de son exécution.

Ensuite, concernant la simulation de la migration : la migration est un processus complexe, dépendant fortement de nombreux facteurs comme le type de VM, sa taille, la taille de lien réseau tant sur la source que sur l'hôte de destination. La topologie réseau ne peut pas être modélisée dans DCWoRMS. C'est pourquoi avec DCWoRMS, bien que nous puissions migrer des tâches entre des noeuds, comme nous le ferions avec des machines virtuelles, nous pouvons seulement modéliser le coût de la migration par une durée.

Finalement, en ce qui concerne le nombre de processeurs/cœurs à être alloués aux tâches. Il est possible d'allouer dans DCWoRMS aussi bien les coeurs que les processeurs, nous avons voulu conserver le même principe qu'avec les RECS et garder les processeurs comme seule ressource de calcul. Cela signifie aussi que nous avons dû définir la consommation

d'un noeud comme :

$$P_h = (P_h^{max} - P_h^{min}) \times load_h + P_h^{min}$$

avec P_h^{max} la consommation maximale de la machine h, et P_h^{min} la consommation lorsque la machine est dans l'état idle.

La méthodologie d'expérimentation utilisée a été la même que précédemment, nous avons juste augmenté l'échelle prise en compte grâce à la simulation. Nous avons mené trois expérimentations pour différentes charges de travail. Dans la première nous évalué les métriques performance et énergie en comparant avec d'autres algorithmes (l'algorithme d'OpenNebula et des algorithmes de la littérature ont été implémentés). Les deux autres expérimentations ont évalué de deux manières la politique de gestion des machines.

Nous avons utilisé la même génération de tâches que pour l'expérimentation réelle. Nous considérons 100 machines chacune ayant 8 processeurs et 16 GB de RAM. P_h^{min} et P_h^{max} ont été choisies respectivement à 70W et 140W (données prises sur le site SPEC Power [SPE14]).

La charge de travail comprend des tâches de durée variant de 500 secondes à 3500 secondes. Les tâches demandent entre 0.25 et 3.75 processeurs et entre 1024 et 7168 MO de RAM. Toutes ces valeurs sont choisies aléatoirement avec une distribution uniforme. L'arrivée de tâches suit une loi de Poisson, choisie pour avoir une charge au sommet de l'expérience à un certain pourcentage. La durée d'expérience couvre de la première arrivée de tâche au départ de la dernière. Au début toutes les machines sont allumées.

Les algorithmes BestFit, SOPVP (c'est à dire une allocation initiale suivant un algorithme de BestFit puis une réallocation par consolidation) et Stripping utilisé par *mm_sched* ont été implémentés dans DCWoRMS. Chaque algorithme utilise la stratégie Pivot pour la gestion des machines avec une politique d'extinction qui est :

$$nb\_actual\_nodes - nb\_expected\_nodes/2$$

Nous avons évalué pour différentes charges prises entre 20% et 80% par pas de 10% la consommation d'énergie et nous avons reproduit chaque expérience 10 fois pour chaque charge et chaque algorithme. Au total, 420 expérimentations ont été réalisées.

Comparaison avec d'autres algorithmes. La figure 3.3 représente la comparaison entre les différents algorithmes. L'algorithme NOHM représente SOPVP sans gestion des machines.

Comme nous pouvions nous y attendre, NOHM consomme presque 3 fois plus d'énergie que SOPVP pour de faibles charges car toutes les machines sont gardées allumées. L'augmentation d'énergie consommée est du à l'overhead induit par l'exécution de tâches sur les noeuds allumés.

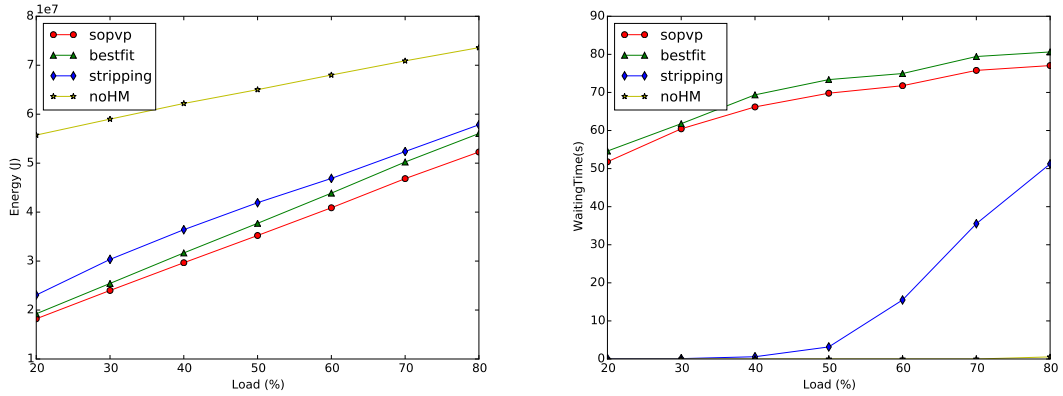


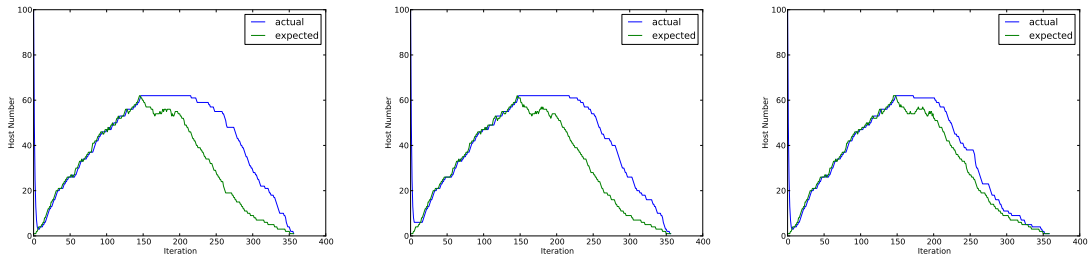
FIGURE 3.3 – Consommation d'énergie et temps d'attente pour différentes charges

Nous pouvons aussi voir que BESTFIT est presque aussi bon comme SOPVP pour de faibles charges, mais se dégrade lorsque la charge augmente et devient aux environs de 6-7 % plus mauvais et se rapproche des performances de STRIPPING. La principale raison est qu'il n'y a pas beaucoup de redistribution à faire lorsque la charge est faible pour diminuer la consommation d'énergie, l'allocation initiale est donc la plus importante dans ce cas. La différence entre STRIPPING et SOPVP commence à 20-21% à charge faible pour finir aux environs de 10% pour des charges plus importantes.

La figure 3.3 compare le temps d'attente moyen pour tous les algorithmes. Comme nous pouvons le voir, sans stratégies de gestion des machines, il n'y a presque aucun temps d'attente puisque toutes les machines sont allumées, le seul temps d'attente qui peut se produire correspond à la situation où le système est plein et où l'ordonnanceur ne trouve pas un placement tout de suite pour les tâches qui arrivent.

L'algorithme STRIPPING, qui était le pire des trois algorithmes en termes de consommation d'énergie, est le meilleur en termes de temps d'attente moyen. La principale raison est qu'il alloue les tâches sur les machines ayant l'utilisation la plus basse, il se comporte comme un algorithme de round-robin (allouant une tâche à chaque noeud) jusqu'à ce qu'une tâche soit placée sur chaque noeud allumé. Au fur et à mesure que l'expérience progresse, la politique de gestion des machines va éteindre des noeuds, ainsi, le nombre de machines utilisées excédera le nombre de machines qui auraient du être allumées, consommant ainsi une trop grande quantité d'énergie. Ceci signifie aussi que puisque nous ne pouvons pas éteindre les machines où une tâche fonctionne, l'algorithme les utilise pour exécuter des tâches diminuant ainsi le temps d'attente.

Il y a une petite différence entre les algorithmes BESTFIT et SOPVP due au fait que



(a) BestFit (b) Stripping (c) SOPVP

FIGURE 3.4 – Différence entre "Actual" et "Expected" pour une charge de 50%

la réallocation de tâches libère de la place pour allouer plus rapidement les tâches qui arrivent. C'est grâce à la réallocation que l'on peut économiser de l'énergie et améliorer la QoS.

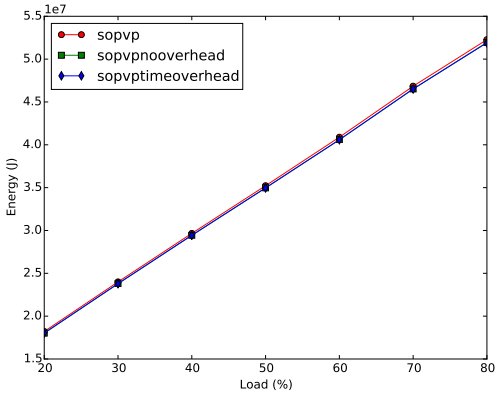
Analyse de la relation entre la gestion des machines et l'algorithme de placement. Nous avons également souhaité analyser par simulation le lien existant entre les stratégies de gestion de machines et les algorithmes de placement.

Il y a une relation forte entre ces deux parties de l'approche quand le but est d'être plus efficace en énergie, en effet, d'une part éteindre puis rallumer des machines les rendra indisponibles pour un temps, empêchant l'allocation de tâches sur ces machines. D'un autre côté si l'algorithme de consolidation choisit un sous-ensemble de machines à utiliser et que la politique de gestion des machines ne les éteint pas alors la réallocation n'aura servi à rien, nous aurons gaspillé des ressources.

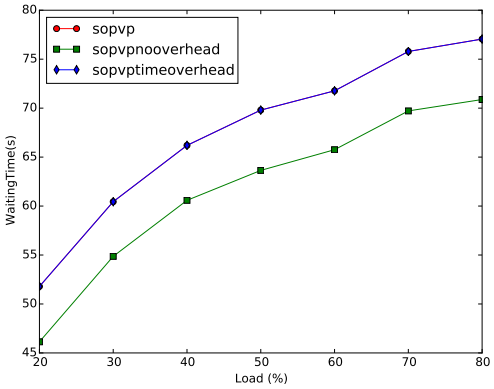
Pour une charge de travail fixée, nous avons comparé à chaque appel du placeur le nombre de machines allumées et le nombre de machines que la politique de gestion des machines avait calculé. Nous avons réalisé cette expérimentation pour les algorithmes *BestFit*, *Stripping* et *Pivot* et nous l'avons reproduite pour différentes charges. Nous ne présentons ici que les résultats pour une charge de 50% car la tendance est la même.

La figure 3.4 montre le nombre de machines allumées à chaque itération (c'est à dire à chaque appel de l'algorithme de placement) (appelées sur les graphes *actual*) ainsi que le nombre de machines que la stratégie de gestion des machines (PIVOT) a calculé comme devant être allumées (appelées sur les graphes *expected*).

Plus la différence entre *actual* et *expected* est importante et plus la stratégie de gestion des machines et le placement coopèrent mal. Si nous regardons la différence entre STRIPPING et SOPVP sur les figures 3.4b et 3.4c nous pouvons constater qu'il n'y a pas de problème à allumer des machines pour atteindre l'objectif donné par PIVOT. Cependant, lorsque



(a) Consommation énergétique pour différentes charges



(b) Temps d'attente pour différentes charges

FIGURE 3.5 – Consommation d'énergie et temps d'attente pour différentes charges

PIVOT demande à éteindre des machines, il se peut que l'on ne trouve pas de machine vide pouvant être éteinte. C'est pourquoi vers la fin de l'expérimentation quand la charge diminue, il y a une différence entre STRIPPING et SOPVP, la réallocation de SOPVP permet d'atteindre plus facilement ou plus rapidement l'objectif de PIVOT.

Impact du surcoût d'allumage et d'extinction. Enfin, nous avons souhaité étudier par la simulation l'impact de l'overhead de l'allumage et de l'extinction des machines sur la consommation d'énergie et le temps d'attente. Pour cela, nous avons implémenté dans DCWoRMS : SOPVPNOOVERHEAD qui correspond à l'algorithme de placement de SOPVP en considérant que l'allumage et l'extinction sont immédiats et SOPVPTIMEOVERHEAD pour lequel l'overhead est uniquement temporel (l'allumage/l'extinction prennent du temps mais ne consomment pas d'énergie).

La figure 3.5a compare pour chaque algorithme l'énergie consommée. SOPVPNOOVERHEAD et SOPVPTIMEOVERHEAD consomment presque autant. Avec SOPVPTIMEOVERHEAD le surcout d'énergie provient du ralentissement. Cependant les deux algorithmes sont très proches. Par contre, la consommation de SOPVP est supérieure aux deux autres.

La figure 3.5b compare le temps d'attente moyen pour chaque algorithme. Cette fois, SOPVPNOOVERHEAD est le meilleur, les deux autres ayant la même performance. En effet SOPVPTIMEOVERHEAD, même en considérant qu'il ne consomme pas d'énergie en démarrant les machines, elles restent pendant un certain temps indisponibles, augmentant ainsi le temps d'attente des tâches de 10%.

En résumé, le surcout induit par l'allumage et l'extinction de machines ne peut être négligé. Cela représente dans les expériences réalisées environ 10s et quelques pourcents de plus d'énergie.

Dans cette partie, nous avons proposé une réallocation dynamique de machines virtuelles prenant en compte tous les surcoûts [BS14]. La simulation a validé les expérimentations en réel même s'il y a des différences. La politique *Pivot* peut être améliorée en intégrant par exemple des techniques de prédiction de la charge.

3.4 Placement avec prise en compte de la propagation de chaleur

Dans le cadre du projet européen CoolEmAll détaillé section 3.5, un modèle du refroidissement et des équipements électriques a été étudié par les partenaires. L'objectif était de calculer la consommation des équipements du refroidissement comme une fonction de la charge, de la température ambiante [FCDCO+15]. Le modèle considère un centre de calcul de taille moyenne avec une unité de refroidissement CRAC *Computer Room Air Conditioner*, ventilateurs, PDU et éclairages. Les pièces du centre de calcul était composées de racks, un rack était composé de RECS et chaque RECS avait un ensemble de noeuds hétérogènes.

Notre plateforme RECS hétérogène est composée de 8 noeuds en Intel i7-2715QE, 4 noeuds d'Atom Intel D510 et 6 noeuds d'AMD G-T40N ayant des configurations hardware différentes. Les 18 noeuds sont disposés dans deux rangées comme le montre la Figure 3.6, où deux noeuds le long de la même colonne partagent une paire d'entrée d'air et de sortie d'air, avec un flux d'air fourni par des ventilateurs.

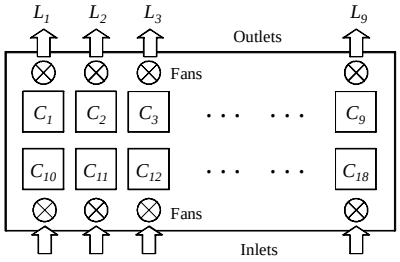


FIGURE 3.6 – Serveur RECS.

Les travaux que nous allons présenter ici ont consisté dans le placement des tâches soumises. Ce travail peut se faire à différentes échelles : au niveau de la pièce d'un datacenter, il est nécessaire de choisir le rack où la tâche sera exécutée puis le RECS puis le noeud

sur lequel l'allocation sera effectuée. L'objectif d'un placement avec prise en compte de la propagation de chaleur sera d'éviter les points chauds : donc de répartir uniformément la chaleur. Intuitivement, on cherchera à placer les noeuds produisant le plus de chaleur en bas des racks où la ventilation est la meilleure. Dans le cadre du projet, nous avons proposé une stratégie "location-aware" pour le choix du rack (puisque l'emplacement physique a son importance) puis une stratégie d'équilibrage de charge à l'intérieur du rack et finalement une stratégie "thermal-aware" pour prendre en compte la température des noeuds [FCDCO+15]. Ceci peut être combiné avec des leviers verts comme le DVFS (en effet le on/off des noeuds ou le suspend to ram auraient un impact sur la redistribution de la chaleur dans le rack/la pièce).

En plus de l'objectif traditionnel de performance pour l'application, nous considérons aussi la consommation d'énergie des serveurs et le déséquilibre thermique comme des objectifs supplémentaires. Les études ont montré que la chaleur a des impacts directs sur l'efficacité et entraîne des coûts de refroidissement non négligeables dans les datacenters [MCRS05, TGV08]. Comme ces objectifs peuvent être contradictoires, le but est d'explorer efficacement leurs différences et si possible, optimiser deux critères ou plus simultanément. Nous appliquons deux approches complémentaires : le placement des applications et le placement de machines.

En effet, en présence de serveurs hétérogènes, le placement des machines peut influencer la distribution de la chaleur produite. Comme il n'est pas possible de changer dynamiquement la configuration physique du datacenter en fonction de la charge des machines, nous proposons un placement de machine statique et proposons une heuristique qui minimise la température maximale de sortie des serveurs en utilisant des caractéristiques de distribution de chaleur dans l'état idle. Nous allons donc, dans ce qui suit présenter différentes approches de prise en compte de la propagation de chaleur à travers différents algorithmes de placement d'applications. Nous terminerons par une proposition d'algorithme de placement statique de machines physiques.

3.4.1 Heuristiques avec contrainte de température

Dans une première approche de la prise en compte de la température dans le placement de tâches, nous avons proposé deux modèles : un modèle temporel de la température et un modèle spatial. Nous formalisons le problème de placement comme un placement avec des contraintes à respecter. Nous proposons une heuristique prenant en compte la performance et la température en proposant un équilibrage de charge avec contrainte de seuil de température.

Modèle temporel de température

Nous avons appliqué le modèle *lumped RC model* [SSH+03] pour caractériser le comportement temporel de la température d'un noeud de calcul. Ce modèle est souvent utilisé comme première approximation du comportement thermique d'un processeur. Il est basé sur le lien existant entre le transfert de chaleur et le circuit électrique *Resistance-Capacitance*. Des modèles plus complexes existent basés sur la dynamique des fluides *Computational Fluid Dynamics*. Néanmoins, le modèle *lumped RC model* tout en étant simple a de bons résultats. La température du processeur suit la loi de Newton du refroidissement : le refroidissement est proportionnel à la différence entre la température du device et celle de la température ambiante de l'environnement. La chaleur conduisant à l'augmentation de la température du processeur est elle proportionnelle à la consommation de puissance du processeur.

Ainsi, la température d'un noeud à l'instant t est une combinaison linéaire de sa température à l'instant $t-1$, de la puissance consommée par le noeud et de la température de l'arrivée d'air dans l'intervalle $(t-1, t]$.

En notant $T_i(t)$ la température d'une machine M_i à l'instant t, R et C la résistance et la capacité thermique, $P_i(t)$ la consommation de puissance, $T_i^{in}(t)$ la température de l'arrivée d'air ; en définissant $f = e^{-\frac{1}{RC}}$ et en supposant que la température de l'arrivée d'air et la consommation de puissance sont constantes dans l'intervalle $(t-1, t]$, nous pouvons définir l'équation du modèle temporel :

$$T_i(t) = (1-f) \cdot \Big(P_i(t) \cdot R + T_i^{in}(t)\Big) + f \cdot T_i(t-1) \ . \tag{3.1}$$

Ce modèle est valide au niveau d'un processeur. Dans un environnement multi-processeurs, il faut prendre en considération les interférences entre processeurs en prenant en compte la circulation de chaleur entre processeurs voisins. C'est que nous appelons le comportement spatial.

Modèle spatial de température

La température d'arrivée d'air d'un noeud est affectée par ses noeuds voisins. Les centres de calcul sont soumis au phénomène de circulation de chaleur : l'air chaud évacué par la sortie d'air des serveurs recircule dans la pièce, se mélange avec l'air froid fourni par le CRAC. Ainsi, la température d'arrivée d'air des serveurs est supérieure à la température d'air fourni par l'unité de refroidissement. En supposant que le système de refroidissement fournit un flux d'air constant, la chaleur produite par chaque machine peut être décrite comme une *matrice de distribution de chaleur* [TGV08] $\mathbf{D}$ où chaque élément $d_{i,k} \in \mathbf{D}$ représente la fraction de chaleur (l'augmentation de température) sur l'arrivée d'air de la machine M_i généré par M_k. La matrice de distribution de chaleur proposée est adaptée

de la littérature [TGV08], elle est fonction du schéma de circulation d'air (fonction de l'emplacement des objets dans l'espace) et de la vitesse des ventilateurs. La température de l'arrivée d'air de M_i à t peut être représentée par :

$$T_i^{in}(t) = T_{sup} + \sum_{k=1}^{m} d_{i,k} \cdot P_k(t) \ , \tag{3.2}$$

où T_{sup} représente la température fournie par l'unité de refroidissement. T_{sup} est supposée constante. La matrice dépend du schéma de circulation d'air dans le centre de calcul qui est fixé pour une configuration physique donnée.

En combinant les deux modèles on arrive à la conclusion que la température d'un noeud suit une évolution temporelle et une influence de la circulation de chaleur. En changeant la consommation d'un noeud de calcul on va influencer la température de plusieurs noeuds. Les équations ci-dessous évaluent la température d'un noeud à l'instant t :

$$T_i(t) \quad = \quad (1-f) \cdot \left(P_i(t) \cdot R + T_{sup} + \sum_{k=1}^{m} d_{i,k} \cdot P_k(t) \right) + f \cdot T_i(t-1) \ . \tag{3.3}$$

En définissant la température stationnaire d'un noeud comme étant $T_i^{ss}(t) = P_i(t) \cdot R + T_{sup} + \sum_{k=1}^{m} d_{i,k} \cdot P_k(t)$. L'Equation (3.3) devient :

$$T_i(t) = (1-f) \cdot T_i^{ss}(t) + f \cdot T_i(t-1) \ . \tag{3.4}$$

A notre connaissance, ceci est le premier modèle analytique proposé facilitant la proposition d'algorithmes de placement.

Algorithmes proposés

Contrairement aux problèmes classiques de placement prenant en compte le makespan, le problème se complexifie lorsque l'on prend en compte la température car il devient nécessaire de considérer les deux dimensions : temporelles et spatiales. En prenant en compte le modèle temporel on peut être amené à empêcher l'exécution à vitesse maximale du processeur pour une tâche (au moins à certains intervalles de temps) ceci afin d'éviter la violation du seuil thermique. En considérant le modèle spatial, une exécution de tâche peut aussi être limitée par l'interférence des autres noeuds à cause de la redistribution de la chaleur. Ceci exige que les décisions de planification locales soient faites avec une perspective plus globale.

Pour répondre à ces problématiques, nous complétons la politique d'équilibrage de charge traditionnelle par une gestion de la température qui utilise DVFS, ne prend pas en compte la migration mais peut introduire des périodes de temps "idle" pour la tâche (on suspend l'exécution de la tâche) pour éviter que la température d'un noeud ne dépasse un seuil maximal. L'heuristique proposée est donc basée sur deux principes :

— *Equilibrage de charge* : permet de décider sur quel noeud on alloue la tâche. Chaque tâche sera affectée à une file $queue_i$ pour chaque machine M_i.
— *Gestion de la température* : permet de décider à quelle vitesse le noeud exécutera la tâche à chaque pas de temps pour respecter un seuil de température maximum.

Nous notons : $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ les n tâches à allouer sur $\mathcal{M} = \{M_1, M_2, \cdots, M_m\}$ de m machines physiques (qui dans cette partie seront considérés comme homogènes). Chaque machine a une consommation statique de puissance P_{static} lorsqu'aucune tâche ne s'exécute. Chaque tâche $J_j \in \mathcal{J}$ a une date d'arrivée *release time* r_j, un temps de calcul *processing time* w_j et une consommation dynamique de puissance p_j. w_j et p_j sont considérés connus suite à un profilage des applications. Nous nous plaçons dans le contexte HPC, un noeud exécute une seule tâche à la fois. Nous cherchons à exploiter le mécanisme DVFS des processeurs, nous avons un ensemble de vitesses utilisables $speed = \{s_0, s_1, \cdots, s_{max}\}$ avec $s_0 = 0$ qui correspond au cas où le processeur est dans l'état idle. $x_{ij}(t) \in \{0, 1\}$ est une variable binaire qui vaut 1 si la tâche J_j s'exécute sur la M_i pendant l'intervalle de temps $(t-1, t)$, 0 sinon. $s_i(t)$ représente la vitesse du noeud M_i à l'instant t (nous supposerons la vitesse fixe pendant chaque intervalle de temps mais pouvant varier d'un intervalle à l'autre). La consommation totale d'un noeud M_i est la somme de sa consommation statique et de sa consommation dynamique :

$$P_i(t) = P_{static} + \sum_{j=1}^{n} x_{ij}(t) \cdot s_i(t)^{\alpha} \cdot p_j \tag{3.5}$$

La date de fin d'une tâche *completion time* est notée c_j ; nous allons chercher à minimiser le *makespan* qui correspond au maximum des dates de fin de tâche $C_{max} = \max_{j=1\cdots n} c_j$ avec un *seuil de température* T_{thresh} pour tous les noeuds : $T_i(t) \leq T_{thresh}$ avec $T_i(t)$ calculé grâce au modèle spatio-temporel. En plus du makespan nous considérons donc également la consommation d'énergie des tâches :

$$E = \int_0^t P_i(t)dt$$

Le makespan est notre premier objectif et l'énergie le second, comme dans les travaux de [FKLB08, LZ09].

Etant donné que la température du système de refroidissement et la puissance statique ont un impact sur la température qui ne varie pas au cours du temps, nous supposons que $T_{sup} = 0$ et $P_{static} = 0$. Nous supposons la consommation de puissance d'une *tâche* inférieure à P_i^{peak} consommation de puissance maximale d'un serveur M_i : puissance maximale autorisée afin de ne pas dépasser le seuil de température.

$$P_i^{peak} = \frac{T_{thresh}}{(1-f) \cdot (R + d_{i,i})} \ . \tag{3.6}$$

De même la température d'un noeud est supposée ne pas dépasser le seuil à cause de la redistribution de chaleur à condition que la consommation de chaque serveur ne dépasse pas leur puissance maximale respective. Pour chaque serveur M_i à l'instant t, nous avons : $\sum_{k=1}^{m} d_{i,k} \cdot P_k(t) \leq T_{thresh}$ si $P_k(t) \leq P_k^{peak}$ pour tout $1 \leq k \leq m$. Ce qui implique pour chaque noeud M_i :

$$\sum_{k=1}^{m} \frac{d_{i,k}}{(1-f) \cdot (R + d_{k,k})} \leq 1 \ . \tag{3.7}$$

Pour définir l'heuristique de placement nous introduisons :

- P_i^{crit} la puissance critique d'un *noeud* c'est à dire la puissance pouvant être consommée sans dépasser le seuil de température. $P_i^{crit} = \frac{T_{thresh}}{R+d_{i,i}}$.
- s_{ij}^{crit} la vitesse critique pour une tâche J_j s'exécutant sur un noeud M_i comme étant la plus grande vitesse possible qui est inférieure ou égale à $\sqrt[\alpha]{\frac{P_i^{crit}}{p_j}}$ avec p_j la consommation de puissance d'une tâche à vitesse maximale du job J_j.
- $w_{ij}^T(t)$ le *thermal-aware work* d'une tâche J_j sur un noeud M_i défini par $w_{ij}^T(t) = \frac{w_j(t)}{s_{ij}^{crit}}$ où $w_j(t)$ représente le temps de calcul restant de la tâche à l'instant t.
- $L_i^T(t)$ la *thermal-aware load* d'un noeud M_i est la somme des *thermal-aware work* de toutes les tâches de la file d'attente du noeud c'est à dire

$$L_i^T(t) = \sum_{J_j \in \mathcal{Q}_i} w_{ij}^T(t)$$

- $\widehat{T}_i(t)$ la *temperature slack* d'un noeud M_i à t est la température stationnaire maximum autorisée sur ce noeud pour ce pas de temps. La *temperature slack* pour le prochain pas de temps est calculée conformément à l'Equation (3.4) par

$$\widehat{T}_i(t+1) = \frac{T_{thresh} - f \cdot T_i(t)}{1-f} \tag{3.8}$$

- $\widehat{P}_i(t)$ la *power slack* d'un noeud M_i à t est la puissance consommée maximale autorisée pour ce pas de temps.

L'algorithme 3 présente le pseudocode de notre heuristique qui est invoquée à chaque pas de temps. L'algorithme fonctionne en deux étapes : il équilibre tout d'abord la charge des tâches arrivées dans l'intervalle de temps puis utilise le levier DVFS pour régler la vitesse des processeurs afin que la température des noeuds soient en dessous du seuil critique.

Pour la répartition de charge, si de nouvelles tâches sont arrivées pendant le pas de temps précédent, l'heuristique assigne chaque nouvelle tâche à un serveur qui aboutit à la plus petite charge "thermal-aware" (lignes 1-12). Ensuite, la partie thermique de l'heuristique régule les températures des serveurs en leur assignant une vitesse appropriée en utilisant

DVFS. Les serveurs sont triés selon leurs charges "thermal-aware" actuelles : celui ayant la charge la plus élevée a la priorité la plus haute (ligne 13). En cas de conflit, l'heuristique maintiendra une vitesse élevée pour les serveurs avec la priorité la plus haute (qui ont donc aussi la charge la plus importante) en réduisant la vitesse de serveurs de basse priorité. Intuitivement, ceci fournit la meilleure répartition de charge dynamique pendant le temps d'exécution afin de minimiser le makespan. Pour chaque noeud M_i, l'algorithme calcule sa *temperature slack* $\widehat{T}_i(t+1)$ pour le prochain pas de temps $t+1$ (lignes 14-16). Les serveurs sont ensuite considérés un par un, leur *power slack* $\widehat{P}_i(t+1)$ est calculée pour chaque serveur M_i (ligne 19) en se basant sur la *temperature slack* du serveur et de celles des serveurs de priorité élevée $M_1, \cdots, M_{i-1}$ pour lesquels on a calculé une vitesse. Les serveurs de priorité plus faible $M_{i+1}, \cdots, M_m$ ne sont pas considérés car on n'a pas encore calculé leur vitesse. La première tâche dans la file d'un serveur sera exécutée avec une vitesse de processeur maximale garantissant que la consommation de puissance résultante n'excédera pas le *power slack* (ligne 20). Finalement, les *temperature slack* sont mises à jour en considérant la contribution de température au serveur lui-même (ligne 21) et à tous les serveurs par la recirculation de chaleur (lignes 22-24).

Il faut bien noter que l'algorithme peut aussi s'appliquer sans DVFS. Dans ce cas, l'ensemble des vitesses de processeurs possible ne contiendrait que deux valeurs : 0 et 1. Une tâche serait donc soit exécutée à vitesse maximale soit suspendue.

Evaluation

L'évaluation de nos propositions s'est faite par simulation (simulateur écrit en matlab). Nous avons simulé 50 serveurs (homogènes pour simplifier le problème) avec une distribution de chaleur issue de [TGV08]. Les serveurs ont comme dans [KGWB08] 5 vitesses de processeur possibles {0, 0.6, 0.733, 0.866, 1}. Nous avons paramétré les différentes équations avec les articles [BBS+00], [ZC07], [SSH+03], ainsi dans notre simulation $f = 0.5$ et $R = 0.7$. Conformément aux préconisations de l'ASHRAE (Association américaine des professionnels du conditionnement d'air) (`http://tc99.ashraetcs.org/`), la température fournie par le système de refroidissement est $T_{sup} = 25°C$. La puissance statique de chaque serveur est supposée contribuer de 10°C à chaque serveur, dont la puce a une température de jonction de 95°C (température locale de la zone de silicium qui forme la jonction sur un composant électronique ; la relation entre la température de jonction et la température externe du composant est donnée par la loi d'Ohm thermique. On trouve dans la littérature différentes valeurs de température de jonction comprises entre 85°C et 100°C [EJF14]).

Ainsi la température seuil est fixée à $T_{thresh} = 95°C - 25°C - 10°C = 60°C$. Le temps de calcul des tâches suit une loi uniforme (de 0 à quelques heures). La consommation de puissance dynamique des tâches suit également une loi de distribution uniforme entre $(0, P_{peak})$. Les résultats présentés sont des moyennes de 50 simulations.

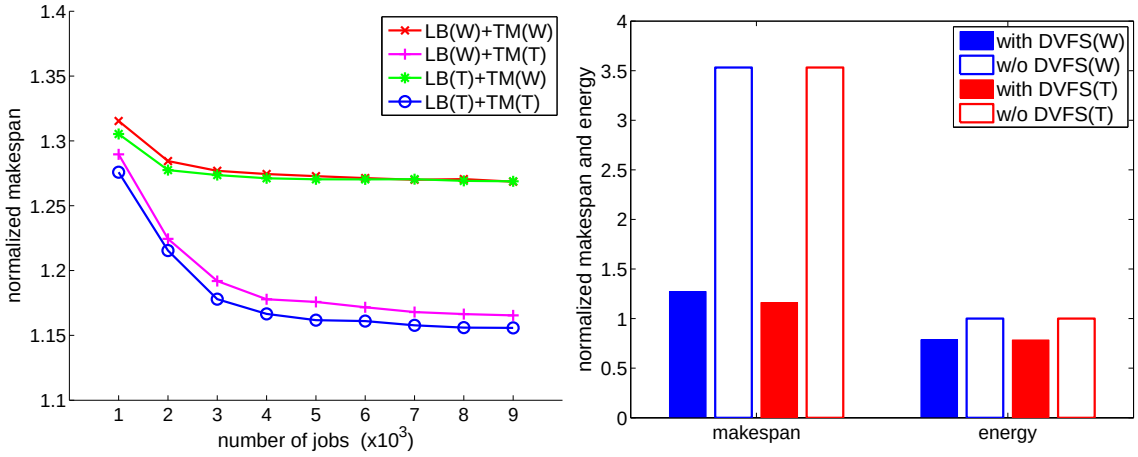
Input: local queue $\mathcal{Q}_i$, temperature $T_i(t)$ and thermal-aware load $L_i^T(t)$ of each server $M_i \in \mathcal{M}$ at time t, and heat-distribution matrix $\mathbf{D}$.
Output: load balancing and thermal management decisions for time step $t+1$.

if new jobs arrive in $(t-1, t]$ **then**
  for each arriving job J_j **do**
    $i' = 0$ and $L_{min}^T = \infty$
    for $i = 1$ to m **do**
      find largest speed $s_{ij}^{crit} \in \mathcal{S}$ that satisfies $s_{ij}^{crit} \leq \sqrt[\alpha]{\frac{P_i^{crit}}{p_j}}$ and let $w_{ij}^T = \frac{w_j}{s_{ij}^{crit}}$
      if $L_i^T(t) + w_{ij}^T < L_{min}^T$ **then**
        $L_{min}^T = L_i^T(t) + w_{ij}^T$ and $i' = i$
      end if
    end for
    assign job J_j to node $M_{i'}$ and update $L_{i'}^T(t) = L_{i'}^T(t) + w_{i'j}^T$
  end for
end if
sort (and rename) the servers in non-increasing order of thermal-aware load, i.e., $L_1^T(t) \geq L_2^T(t) \geq \cdots \geq L_m^T(t)$
for $i = 1$ to m **do**
  compute $\widehat{T}_i(t+1) = \frac{T_{thresh} - f \cdot T_i(t)}{1-f}$
end for
for $i = 1$ to m **do**
  if $\mathcal{Q}_i \neq \emptyset$ **then**
    compute $\widehat{P}_i(t+1) = \min\left(\frac{\widehat{T}_1(t+1)}{d_{1,i}}, \frac{\widehat{T}_2(t+1)}{d_{2,i}}, \cdots, \frac{\widehat{T}_{i-1}(t+1)}{d_{i-1,i}}, \frac{\widehat{T}_i(t+1)}{R+d_{i,i}}\right)$
    find largest speed $s_{ij} \in \mathcal{S}$ that satisfies $s_{ij} \leq \sqrt[\alpha]{\frac{\widehat{P}_i(t+1)}{p_j}}$, where p_j is the full-speed power consumption of the first job $J_j \in \mathcal{Q}_i$, and set $s_i(t+1) = s_{ij}$
    update $\widehat{T}_i(t+1) = \widehat{T}_i(t+1) - p_j \cdot s_{ij}^\alpha \cdot R$
    for $k = 1$ to m **do**
      update $\widehat{T}_k(t+1) = \widehat{T}_k(t+1) - p_j \cdot s_{ij}^\alpha \cdot d_{k,i}$
    end for
  end if
end for

Algorithm 3: Heurisitique de placement "Thermal-Aware" appelée à l'instant t

Comparaison de performance des heuristiques. Pour évaluer la performance de l'algorithme proposé, nous avons étudié l'impact des deux parties de l'algorithme : celle de l'équilibrage de charge et celle de la gestion de la température. Nous avons comparé quatre heuristiques notées LB(W)+TM(W), LB(W)+TM(T), LB(T)+TM(W), LB(T)+TM(T), en fonction des différentes définitions de charge :

— W représente la définition classique de la charge que l'on trouve dans la littérature
— T représente la "thermal load" introduite dans ces travaux.



(a) Makespan (b) DVFS

FIGURE 3.7 – (a) Comparaison du Makespan pour les différentes heuristiques. (b) Impact du DVFS sur la performance des heuristiques.

La figure 3.7a présente les résultats de simulation pour un nombre de tâches allant de 1000 à 9000 (ce qui correspond en moyenne à un nombre moyen de tâches par serveur entre 20 et 180). Le makespan est normalisé avec $\frac{1}{m}\sum_{j=1}^{n} w_j$. Nous pouvons voir que le makespan de l'heuristique "thermal-aware" LB(T) +TM (T) s'améliore avec la charge du système, tandis que l'heuristique LB(W) +TM (W) est moins sensible à la charge. En particulier, l'heuristique "thermal-aware" permet jusqu'à 10% d'amélioration pour des charges moyennes et élevées dont la plupart vient de la prise en compte thermique. Les résultats confirment l'importance d'utiliser le bon indicateur de charge en prenant en particulier des décisions de placement prenant en compte la température.

Impact du DVFS. Nous avons étudié l'impact du DVFS sur le makespan et la consommation énergétique des heuristiques. Seule la consommation énergétique dynamique est prise en compte puisque la consommation statique est la même pour toutes les heuristiques. L'énergie est normalisée par la consommation maximale possible $\sum_{j=1}^{n} w_j p_j$ de toutes les tâches s'exécutant à vitesse maximale.

La figure 3.7b présente les résultats pour une charge moyenne (5000 tâches) pour l'heuristique notée W (charge au sens classique) et l'heuristique T ("thermal-load"). Pour les deux heuristiques, le DFVS a amélioré le makespan de plus de 65% et a amélioré la consommation énergétique de 20%. Contrairement à beaucoup de problèmes d'allocation où l'utilisation du DVFS aboutit à un compromis entre la performance et l'énergie, l'allocation "thermal-aware" profite du DFVS sur les deux plans. En effet, en réduisant les vitesses du processeur,

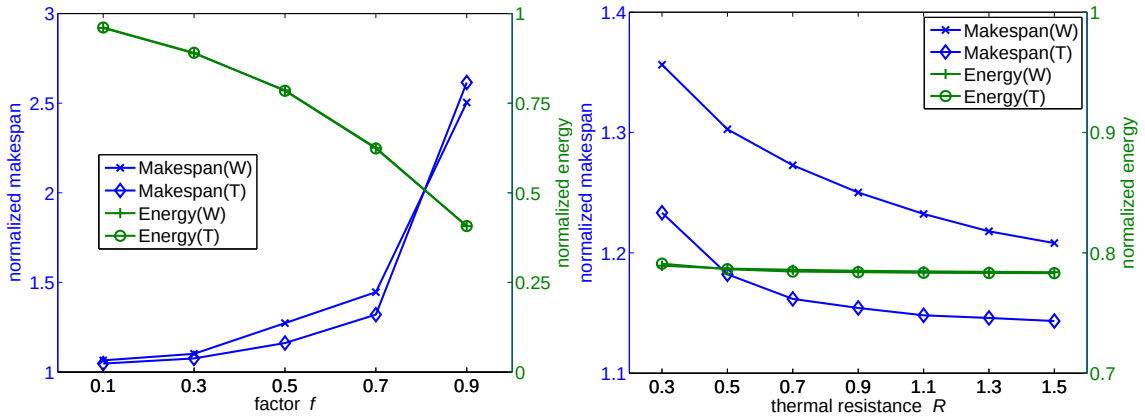


FIGURE 3.8 – Impact du (a) facteur f et (b) de R sur la performance des heuristiques.

le DVFS permet aux noeuds de calcul de respecter leur seuil thermique, de diminuer la consommation d'énergie tout en faisant progresser l'exécution des tâches.

Impact des paramètres thermiques. Pour finir, nous avons évalué l'impact des paramètres thermiques sur la performance des heuristiques en faisant varier f de 0.1 à 0.9 et R de 0.3 à 1.5. La consommation de puissance des tâches change en conséquence (à travers l'équation 3.6).

La Figure 3.8 montre les résultats. L'augmentation de f entraîne une augmentation de la consommation énergétique des tâches, qui renforce l'effet d'interférence des serveurs et par conséquent augmente le makespan. D'autre part, en augmentant la résistance thermique R on réduit la puissance, ce qui affaiblit l'interférence et améliore ainsi le makespan. L'écart du makespan entre les deux heuristiques est relativement stable à part pour les très grandes valeurs de f, auquel cas l'interférence devient si forte que la "thermal-load" seule n'est pas suffisante pour représenter la priorité des serveurs, comme cela est fait dans l'heuristique "thermal-aware". Cependant, puisque les températures des serveurs répondent normalement rapidement à leurs consommations énergétiques, cette situation ne va probablement pas être observée en pratique. Finalement, l'énergie consommée par les deux heuristiques est semblable, les profils de puissance des tâches sont déterminants (mais donnés par les paramètres thermiques) tout en étant indépendants de l'heuristique d'allocation.

En résumé, nous avons proposé une heuristique basée sur la notion de charge thermique ("thermal-aware") et nous avons montré son intérêt par rapport à une prise en compte classique de la charge. Les aspects thermiques sont introduits par la notion de température seuil au niveau de l'entrée d'air des serveurs. Les résultats confirment également l'intérêt

d'utiliser le DVFS dans ce contexte.

En perspective de ce travail, il reste à comparer les performances de notre approche par rapport à des algorithmes de la littérature. Les deux points forts de l'approche sont le modèle spatio-temporel et la notion de *thermal load* qui permet d'intégrer la notion de température dans l'équilibrage de charge et d'éviter ainsi les points chauds. Cette approche se distingue sur deux points des solutions matérielles de constructeurs de la famille de processeurs Sandy Bridge d'Intel qui offre la technologie de power clamping/capping permettant de définir des bornes de puissance pour le processeur : la première différence est que notre approche est pro-active dans le sens où elle permet d'anticiper les points chauds en allouant les tâches en tenant compte de la chaleur. La technologie de power clamping/capping est réactive. La deuxième différence est que notre approche tient compte du critère performance en plaçant les tâches.

Nous allons dans la suite montrer une méthodologie de prise en compte de la chaleur en étudiant un problème d'optimisation multi-critères mais aussi en considérant à la fois la consommation énergétique du calcul et du refroidissement.

3.4.2 Heuristiques avec optimisation multi-critères

Dans [SSPDC14b], nous avons formalisé le problème en cherchant à optimiser le placement de tâches avec plusieurs critères conflictuels. Les objectifs pris en compte sont :

- la performance des tâches. Pour cela, nous avons choisi le temps de réponse moyen des tâches comme métrique définie par $R_{ave} = \frac{1}{n}\sum_{i=1}^{n}(c_i - r_i)$ où c_i est la date de fin de la tâche J_i et r_i la date d'arrivée.
- la consommation énergétique des serveurs. Nous ne prenons en compte que la partie dynamique de la consommation d'énergie des serveurs.
- la température des sorties d'air l des serveurs. Pour assurer un équilibre thermique, nous utilisons deux métriques la température maximale T_{max}^{out} et la température moyenne T_{ave}^{out} des sorties d'air des serveurs. Elles sont définies par

$$T_{max}^{out} = \max_{t_1 \leq t \leq t_2} \max_{1 \leq k \leq l} T_k^{out}(t) \ , \tag{3.9}$$

$$T_{ave}^{out} = \frac{1}{(t_2 - t_1) \cdot l} \int_{t_1}^{t_2} \sum_{k=1}^{l} T_k^{out}(t) dt \ , \tag{3.10}$$

  où $[t_1, t_2]$ représente l'intervalle pendant lesquel des tâches sont arrivées et ont terminé leur exécution. En supposant une température constante T^{in} pour toutes les entrées d'air du serveur (inlet), la température à la sortie L_k est donnée par l'équation :

$$T_k^{out}(t) = T^{in} + g(U_k^{out}(t)) \ , \tag{3.11}$$

où la fonction $g(U)$ [TGV08] convertit la chaleur U (en $Watt$) reçue par la sortie d'air en degrés (augmentation de sa température) ; elle est fonction de plusieurs paramètres ; on pourra se reporter aux travaux cités pour plus de précisions.

L'algorithme 4 présente l'heuristique proposée : il s'agit d'un algorithme glouton cherchant à minimiser $H_{i,j}$ la fonction de coût évaluée lors de l'affectation de la tâche J_i à la machine M_j. En fonction de l'objectif souhaité, $H_{i,j}$ peut être une fonction du temps de réponse, de la consommation d'énergie et de la température de la sortie d'air. La tâche sera allouée sur la machine ayant le coût le plus faible.

Input: Arrival of a new job J_i, and its processing time $P_{i,j}$ and dynamic power consumption $U_{i,j}$ if it is executed on any machine $M_j \in \mathcal{M}$.
Output: Assignment of job J_i to a machine in $\mathcal{M}$.

```
1: j' = 0 and H_{i,j'} = ∞
2: for j = 1 to m do
3:    if H_{i,j} < H_{i,j'} then
4:       H_{i,j'} = H_{i,j} and j' = j
5:    end if
6: end for
7: Assign job J_i to machine M_{j'}
```

Algorithm 4: Algorithme glouton en ligne

Les fonctions de coût des objectifs considérés dans nos travaux [SSPDC14b] sont les suivants :

— *Fastest* : la tâche J_i est allouée à la machine fournissant le temps de réponse minimum.

$$H_{i,j}^{R} = \max(r_i, t_j^{avail}) + P_{i,j} \ , \tag{3.12}$$

où r_i est la date de soumission de la tâche J_i, t_j^{avail} la date à laquelle la machine M_j est disponible et $P_{i,j}$ le temps de calcul.

— *Greenest* : la tâche J_i est allouée à la machine qui génère la consommation d'énergie dynamique minimum.

— *Coolest* : la tâche J_i est allouée à la machine qui minimise la température maximale de la sortie d'air

$$H_{i,j}^{T} = \max_{k=1\cdots l}(T_k^{out} + g(d_{j,k} \cdot U_{i,j})) \ , \tag{3.13}$$

où T_k^{out} est la température de la sortie d'air L_k avant l'affectation de la tâche et $d_{j,k}$ est la fraction de chaleur générée par la machine M_j à la sortie d'air L_k (matrice de distribution de chaleur).

Pour résoudre l'optimisation multi-critères nous avons proposé une approche appelée *fuzzy-based priority* utilisant la fonction de coût composite ci-dessous :

$$H_{i,j}^{X,Y} = \langle \overline{H}_{i,j}^{X}(f), H_{i,j}^{Y} \rangle \ . \tag{3.14}$$

Dans ce cas, les objectifs X et Y sont considérés l'un après l'autre en sélectionnant d'abord les machines offrant la meilleure performance en terme de X, puis en sélectionnant dans cet ensemble de machines celle offrant la meilleure performance en terme de Y. Un facteur *fuzzy factor* $f \in [0, 1]$ est utilisé pour relâcher le critère de sélection sur le premier objectif tout en ayant une marge acceptable. L'introduction de ce facteur permet d'explorer une amélioration du critère Y tout en maintenant la performance sur le critère X dans une marge acceptable (en pourcentage). L'approche sera efficace si un léger compromis avec X peut entraîner une amélioration sur Y. Sachant que si $f = 0$ alors il n'y a aucun compromis et si $f = 1$ alors X n'a aucune importance dans l'optimisation. L'implémentation dans l'algorithme 4 nécessite que la fonction de coût du premier objectif soit normalisé entre 0 et 1 de manière à prendre en compte le *fuzzy factor*.

$$\overline{H}^X_{i,j} = \frac{H^X_{i,j} - H^X_{i,min}}{H^X_{i,max} - H^X_{i,min}} \ , \tag{3.15}$$

où $H^X_{i,min}$ et $H^X_{i,max}$ sont le minimum et le maximum pour X parmi toutes les machines possibles pour la tâche J_i. La comparaison des coûts entre deux machines suit la règle suivante :

Règle de priorité entre deux objectifs : Les coûts engendrés par une tâche J_i sur deux machines M_{j_1} et M_{j_2} satisfont $H^{X,Y}_{i,j_1} < H^{X,Y}_{i,j_2}$ si et seulement si une des règles ci-dessous est respectée :

- $\overline{H}^X_{i,j_1} \leq f < \overline{H}^X_{i,j_2}$, ou
- $\overline{H}^X_{i,j_1} \leq f$ et $\overline{H}^X_{i,j_2} \leq f$ et $H^Y_{i,j_1} < H^Y_{i,j_2}$, ou
- $\overline{H}^X_{i,j_1} < \overline{H}^X_{i,j_2} \leq f$ et $H^Y_{i,j_1} = H^Y_{i,j_2}$, ou
- $f < \overline{H}^X_{i,j_1} < \overline{H}^X_{i,j_2}$, ou
- $f < \overline{H}^X_{i,j_1} = \overline{H}^X_{i,j_2}$ et $H^Y_{i,j_1} < H^Y_{i,j_2}$.

Lorsque plus de deux objectifs doivent être considérés nous avons étudié différentes possibilités : dans [SSPDC14b] nous avons considéré l'optimisation d'un objectif après un autre avec l'approche *fuzzy-based* mais avec le deuxième objectif qui est combinaison d'un ensemble d'objectifs ; par exemple en combinant de manière pondérée les objectifs temps de réponse et consommation d'énergie comme un seul objectif :

$$H^{RE}_{i,j} = \alpha \overline{H}^R_{i,j} + (1 - \alpha)\overline{H}^E_{i,j} \ , \tag{3.16}$$

où $\alpha \in [0, 1]$ représente le poids des critères, cela nécessite bien sûr une normalisation. Ensuite, des fonctions de coûts composites peuvent être construites :

$$H^{T,RE}_{i,j} = \langle \overline{H}^T_{i,j}(f), H^{RE}_{i,j} \rangle \ , \tag{3.17}$$

ou

$$H^{RE,T}_{i,j} = \langle \overline{H}^{RE}_{i,j}(f), H^T_{i,j} \rangle \ . \tag{3.18}$$

Dans [SSPDC14a] nous avons étendu la méthode pour s objectifs :

$$H_{i,j}^{X_1,X_2,\cdots,X_s} = \langle \overline{H}_{i,j}^{X_1}(f_1), \overline{H}_{i,j}^{X_2}(f_2), \cdots, H_{i,j}^{X_s} \rangle \ . \tag{3.19}$$

Discussion relative aux approches de la littérature La figure 3.9 illustre dans le cas d'une optimisation multi-critères avec deux objectifs les approches classiques trouvées dans la littérature.

(1). *Priorité simple*. Considérer une priorité simple entre objectifs est un cas particulier de l'approche proposée qui correspond à $f = 0$. Ceci est utilisé lorsqu'une priorité est imposée entre les objectifs sans compromis.

(2). *Frontière de Pareto*. Cette approche renvoie un ensemble de solutions *non dominées* c'est à dire un ensemble de solutions pour lesquelles il n'y a pas de meilleure performance vis à vis de tous les objectifs. C'est surtout utilisé en mode "hors-ligne" pour évaluer les compromis entre plusieurs objectifs. Dans un contexte de placement en-ligne, il est difficile de maintenir plusieurs solutions, il est alors nécessaire de choisir une solution intermédiaire.

(3). *Optimisation contrainte*. Cette approche optimise un objectif avec des contraintes imposées aux autres objectifs. C'est souvent utilisé lorsque des besoins précis ont été définis : deadline pour les tâches, budget ... L'intérêt de la méthode avec facteur *fuzzy-based* est de spécifier une contrainte relative en terme de pourcentage.

(4). *Somme pondérée*. Cette approche transforme plusieurs objectifs en un seul objectif qui est une combinaison linéaire pondérée. Les différents objectifs doivent être normalisés. La difficulté consiste à choisir de manière appropriée les poids des différents critères.

La priorité *fuzzy-based* est particulièrement adaptée pour les applications HPC dans les centres de calcul où il n'y a pas de contrainte stricte ou de priorité entre la performance et la consommation d'énergie. En comparaison avec l'approche pondérée, la méthode proposée est une alternative intuitive pour décrire les compromis. De plus, comme le montre le schéma Figure 3.9, la solution renvoyée par l'approche *fuzzy-based* se trouve sur la frontière de pareto.

Evaluation Nous avons utilisé le simulateur DCWoRMS dans lequel nous avons modélisé la plate-forme RECS complète *Resource Efficient Cluster Server (RECS)* c'est à dire les 18 noeuds hétérogènes : 8 noeuds Intel i7-2715QE, 4 noeuds Intel Atom D510 et 6 noeuds AMD G-T40N.

La démarche que nous avons suivie a été la suivante : nous avons évalué comment les critères performance, énergie et température pouvaient s'optimiser deux à deux pour différentes valeurs de *fuzzy factor* et pour différentes charges : cela nous a permis de déterminer quels

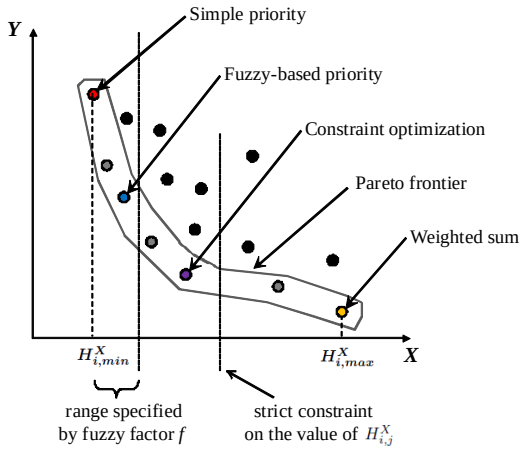


FIGURE 3.9 – Comparaison de l'approche *fuzzy-based* avec d'autres approches d'optimisation bi-objectif. Chaque point représente une solution potentielle. La solution de chaque approche est indiquée.

critères sont antagonistes et quels critères peuvent se minimiser conjointement avec cette méthode. Ensuite, nous avons fixé une fonction de coût composite intégrant les trois critères et nous avons évalué les trois métriques pour différentes valeurs de *fuzzy factor* et de α pour une charge donnée. Cette évaluation a été faite pour différentes charges. Cela nous a permis de calibrer ces deux paramètres d'optimisation. Pour finir, nous avons fixé la valeur du *fuzzy factor* et de α et comparé les performances de notre algorithme par rapport à des algorithmes existants.

Nous ne présentons ici qu'une partie des résultats publiés dans [SSPDC14b]. Les résultats sont une moyenne de dix simulations pour l'approche des fonctions de coûts composites : facteur *fuzzy-based* et poids. La figure 3.12 montre l'impact de la variation du fuzzy facteur f de -1 à 1 quand on minimise deux des trois objectifs pour une charge de 20%. Lorsque $f = -1$, cela signifie que la décision de placement est uniquement basée sur le premier objectif, le deuxième objectif est ignoré. Les figures 3.10a, 3.11b, 3.12c en diagonale montrent l'optimisation pour deux objectifs égaux, cela revient donc à une optimisation mono-objectif. Les figures 3.10b, 3.11a ne montrent pas de changement pour le temps de réponse et la consommation d'énergie dynamique à moins que f soit supérieur à 0.7 pour $H^{R,E}_{i,j}$ et 0.5 pour $H^{E,R}_{i,j}$. Les résultats suggèrent qu'il est difficile d'avoir une bonne performance pour un objectif sans une dégradation considérable sur le deuxième. Etant donné l'importance des deux objectifs, le facteur f peut dans ce cas être calibré entre $[0.6, 0.9]$ pour obtenir un compromis correct. Les figures 3.10c, 3.11c montrent que la température maximale de sortie d'air peut être minimisée en même temps que le temps de réponse

moyen ou la consommation dynamique d'énergie tant que $f \geq 0$ et que $f \leq 0.6$. Des résultats similaires peuvent être observés sur les figures 3.12a, 3.12b où la température est optimisée avant le temps de réponse ou l'énergie.

Les résultats montrent qu'il est difficile de minimiser simultanément l'énergie et le temps de réponse. Nous avons donc combiné les deux objectifs avec $H_{i,j}^{RE}$ défini par l'Equation (3.16) et nous l'avons optimisé avec $H_{i,j}^{T,RE}$ et $H_{i,j}^{RE,T}$ définis dans les Equations (3.17) and (3.18).

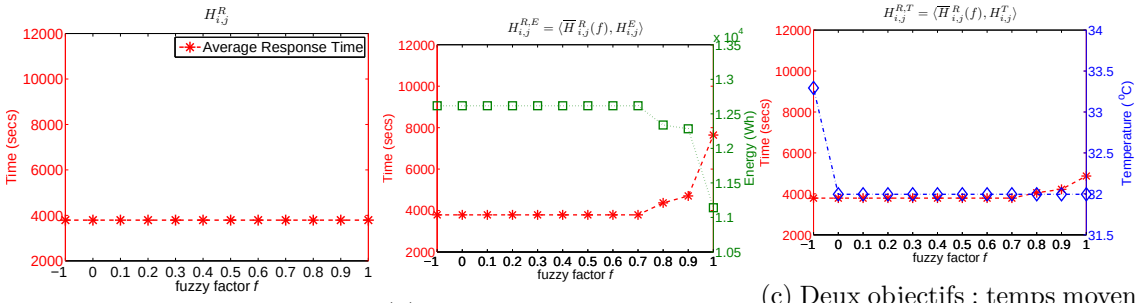
La figure 3.13 montre les résultats pour différentes valeurs de α et f pour une charge de 20%. Lorsque α est grand (signifiant que le temps de réponse est favorisé par rapport à la consommation d'énergie), le temps de réponse est réduit quand f augmente. Le même comportement peut être observé pour la consommation énergétique lorsque α est petit. Ceci démontre l'efficacité d'utiliser f pour améliorer des objectifs combinés. La figure 3.13c montre que la température maximale de la sortie d'air peut être améliorée en considérant le temps de réponse et l'énergie comme second objectif, ce qui correspond aux résultats observés sur les figures 3.12a et 3.12b.

La figure 3.14 montre la performance de l'algorithme appelé *Combined* pour $f = 0.4$ and $\alpha = 0.5$ avec les trois heuristiques *Fastest*, *Greenest* et *Coolest*. Les résultats montrent un bon compromis entre le temps moyen de réponse et la consommation dynamique d'énergie pour différentes charges en utilisant *Fastest* et *Greenest*. De plus, cela maintient une température maximale de la sortie d'air convenable qui est proche de ce qui est obtenu avec l'heuristique *Coolest*.

Dans les travaux [SSPDC14a] nous avons considéré la problématique du coût du refroidissement au niveau du centre de calcul et non plus au niveau d'un serveur. Pour cela, nous avons considéré le coût énergétique complet provenant du calcul et du refroidissement. Le coût du refroidissement est défini par :

$$U^{cool}(t) = \frac{\sum_{j=1}^{m} U_j^{comp}(t)}{\mathrm{CoP}(T^{sup}(t))} \ , \tag{3.20}$$

où CoP est le *coefficient de performance*, défini comme le ratio entre la quantité de chaleur à évacuer divisé par l'énergie nécessaire pour le refroidissement [MCRS05]. Ce coefficient caractérise l'efficacité de l'unité de refroidissement et augmente de manière non-linéaire avec la température fournie par l'unité de refroidissement. Nous avons résolu le problème d'optimisation bi-objectifs (performance et énergie totale) en utilisant la méthode *fuzzy-based*. Nous avons considéré uniquement l'énergie dynamique provenant des serveurs et celle du refroidissement (l'énergie statique du refroidissement est l'énergie consommée pendant les intervalles de temps pendant lesquels aucune tâche n'est soumise). Les résultats ont montré [SSPDC14a] l'efficacité de notre approche pour étudier les compromis entre la consommation d'énergie et la performance tout en prenant en compte le refroidissement.

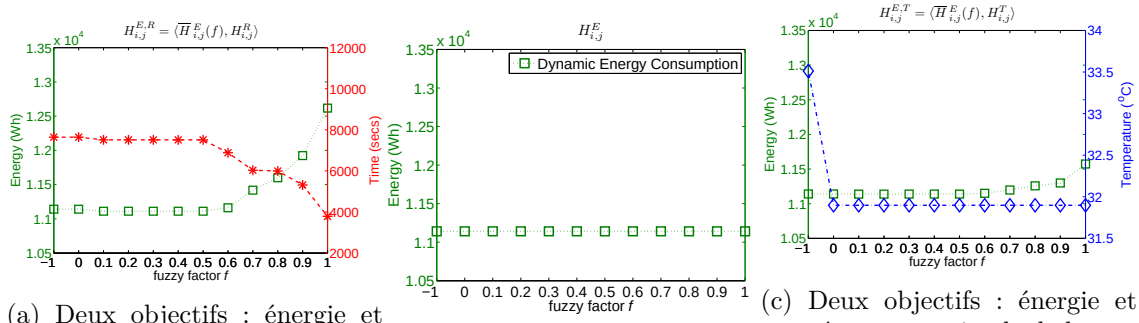


(a) Mono-objectif : temps moyen de réponse

(b) Deux objectifs : temps moyen de réponse et énergie

(c) Deux objectifs : temps moyen de réponse et température maximale de la sortie d'air

FIGURE 3.10 – Utilisation de l'heuristique *fuzzy-based* pour deux objectifs avec une charge de 20%, le temps moyen de réponse est le premier objectif.

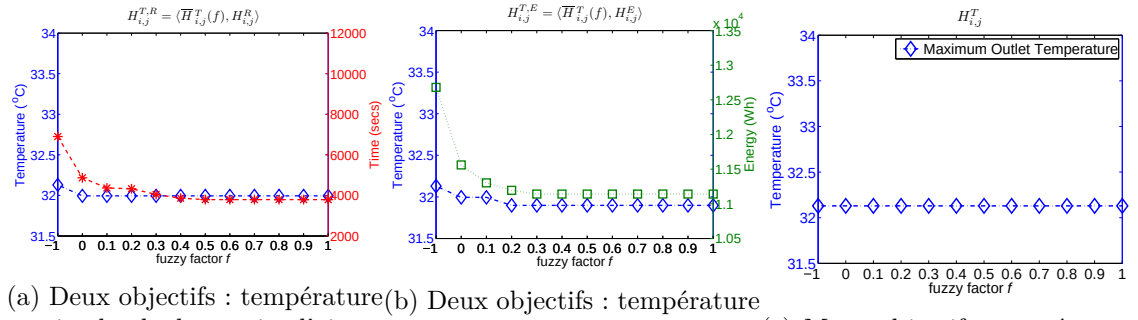


(a) Deux objectifs : énergie et temps moyen de réponse

(b) Mono-objectif : énergie

(c) Deux objectifs : énergie et température maximale de la sortie d'air

FIGURE 3.11 – Utilisation de l'heuristique *fuzzy-based* pour deux objectifs avec une charge de 20%, l'énergie est le premier objectif.

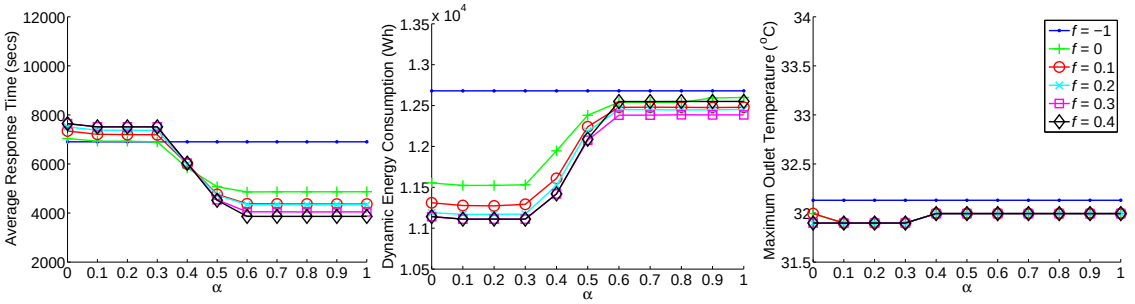


(a) Deux objectifs : température maximale de la sortie d'air et temps moyen de réponse

(b) Deux objectifs : température maximale de la sortie d'air et énergie

(c) Mono-objectif : température maximale de la sortie d'air

FIGURE 3.12 – Utilisation de l'heuristique *fuzzy-based* pour deux objectifs avec une charge de 20%, la température maximale de la sortie d'air est le premier objectif.

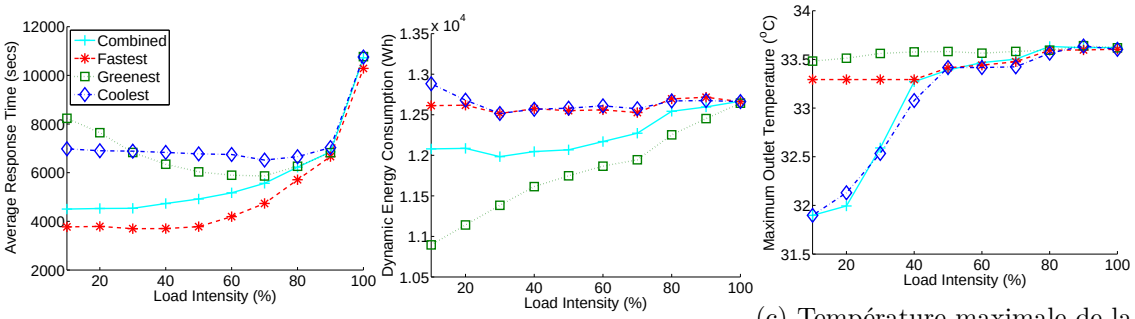


(a) Temps moyen de réponse pour différents f et α

(b) Consommation d'énergie pour différents f et α

(c) Température maximale de la sortie d'air pour différents f et α

FIGURE 3.13 – Optimisation multi-objectif pour $H_{i,j}^{T,RE} = \langle \overline{H}_{i,j}^{T}(f), \alpha \overline{H}_{i,j}^{R} + (1-\alpha)\overline{H}_{i,j}^{E} \rangle$ avec différentes valeurs de f et α pour une charge de 20%.



(a) Temps moyen de réponse

(b) Consommation d'énergie

(c) Température maximale de la sortie d'air

FIGURE 3.14 – Comparaison des heuristiques pour l'optimisation multi-objectif pour $H_{i,j}^{T,RE} = \langle \overline{H}_{i,j}^{T}(f), \alpha \overline{H}_{i,j}^{R} + (1-\alpha)\overline{H}_{i,j}^{E} \rangle$ avec $f = 0.4$ et $\alpha = 0.5$ pour différentes charges.

L'ensemble des validations proviennent de simulation car la prise en compte de la chaleur nécessite des systèmes de grande taille. Néanmoins, les paramètres sont des paramètres réels : les valeurs de la matrice de distribution de chaleur ont été publiées dans [TGV08], le coefficient de performance dans [MC06], les spécifications matérielles proviennent de données publiques [Pas14] ; enfin, les paramètres des *workloads* proviennent de mesures réelles d'applications HPC [DCZJO13], [KOP+13].

3.4.3 Placement statique des serveurs

Lorsque l'on intègre la prise en compte de la chaleur dans un système hétérogène, nous sommes finalement confrontés à deux sous-problèmes. Le premier pour décider du *placement statique des machines* c'est à dire trouver une association de chaque emplacement aux machines ; ensuite, nous devons décider d'une politique de *placement on-line* pour affecter chaque nouvelle tâche à une machine. Nous avons précédemment montré comment nous proposons de mettre en oeuvre le placement de tâches. Le placement de machines virtuelles dans les centres de calcul a donné lieu à de nombreuses publications [BCDCP12, GGQ+13, XF10] mais le placement de machines physiques a lui reçu moins d'attention. Dans [DGLM13], les auteurs ont montré que des clusters apparemment homogènes (les FLOPS (FLoat Operations Per Seconds) des noeuds sont identiques) sont en fait hétérogènes en consommation d'énergie (jusqu'à 22% de différence). Ainsi, même dans des datacenters composés de clusters homogènes les différences de consommations d'énergie justifient de s'interroger sur l'importance du placement des serveurs. D'autre part, le placement de serveurs aura un impact sur le coût de refroidissement de centres de calcul hétérogènes. Les études [TMGC06, TGV08] ont montré que la redistribution de chaleur dans les centres de calcul typiques a les propriétés suivantes :
(1). La position des racks a un impact différent sur la redistribution de chaleur. Typiquement, les serveurs situés en haut "inhalent" plus de chaleur tandis que les serveurs placés en bas contribuent plus à la recirculation de chaleur.

(2). Dans une salle d'informatique fermée avec des emplacements fixes pour tous les objets majeurs, le modèle de flux d'air qui caractérise la recirculation de chaleur est relativement stable.
Tandis que la première propriété suggère que la matrice de distribution de chaleur a tendance à être fortement asymétrique, la deuxième propriété assure que la matrice ne change pas significativement avec des charges de travail différentes. Nous allons exposer dans la suite l'étude réalisée pour le placement des serveurs et montrer l'impact sur le coût du refroidissement.

La contribution ne consiste pas à trouver le meilleur algorithme mais plutôt à prouver l'importance et l'impact du placement des machines physiques sur la température de la sortie d'air.

Nous ne prenons ici en considération que la consommation d'énergie statique. Il y a une corrélation entre la consommation de puissance statique des machines et leur consommation dynamique : une machine avec une consommation statique plus élevée aura une consommation dynamique plus élevée pour l'exécution d'une tâche donnée. Ceci explique que dans cette étude la prise en compte de la consommation statique suffise. Nous avons proposé [SSPDC14b] une heuristique basée sur un algorithme glouton pour réduire la température maximale de la sortie d'air.

L'algorithme 5 résume l'approche. GMP trie les machines par ordre décroissant de consommation d'énergie statique (puisque les machines qui consomment plus de puissance génèreront plus de chaleur), ainsi elles seront placées en premier afin d'éviter des pics de température. Ensuite, chaque machine sera placée dans une position restante qui minimise la température maximale des sorties d'air. La position choisie est ensuite retirée de la liste des emplacements et les températures des sorties d'air sont actualisées.

Input: Set $\mathcal{M} = \{M_1, M_2, \cdots, M_m\}$ of m machines,
set $\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$ of m server positions,
and heat distribution matrix D.
Output: A mapping π from server positions to machines.
1: Sort the machines in descending order of static power consumptions, U_j^{stat} is the static consumption
2: Initialize $T_k^{out} = 0$ for all $1 \leq k \leq l$
3: **for** $j = 1$ to m **do**
4: $x' = 0$ and $T_{max}^{out}(x') = \infty$
5: **for** each $C_x \in \mathcal{C}$ **do**
6: $T_{max}^{out}(x) = \max_{k=1..l} \left(T_k^{out} + g(d_{x,k} \cdot U_j^{stat})\right)$
7: **if** $T_{max}^{out}(x) < T_{max}^{out}(x')$ **then**
8: $T_{max}^{out}(x') = T_{max}^{out}(x)$ and $x' = x$
9: **end if**
10: **end for**
11: Place machine M_j to position $C_{x'}$, i.e., $\pi(x') = j$
12: Update $T_k^{out} = T_k^{out} + g(d_{x',k} \cdot U_j^{stat})$ for all $1 \leq k \leq l$, and update $\mathcal{C} = \mathcal{C} \setminus C_{x'}$
13: **end for**

Algorithm 5: Heuristique gloutonne de placement de machines physiques

L'algorithme a été évalué en le comparant à des variantes de placement dans l'objectif de montrer l'importance de ce placement de serveurs. MP2 trie les machines dans l'ordre croissant et MP3 affecte les machines à la position restante permettant de maximiser la température de la sortie d'air. Ils peuvent paraître contre-productif mais servent uniquement à démontrer l'importance d'un choix pertinent. Les figures 3.15 et 3.16 montrent les trois heuristiques avec la fonction de coût combiné $H_{i,j}^{RE,T}$ avec $f = 0.1$ et $\alpha = 0.4$. Les résultats montrent clairement que le temps de réponse et la consommation d'énergie ne sont pas affectés par différents placement de serveurs. En revanche, GMP permet de réduire la température de la sortie d'air entre 1^oC et 3^oC. Cette différence a un impact sur la

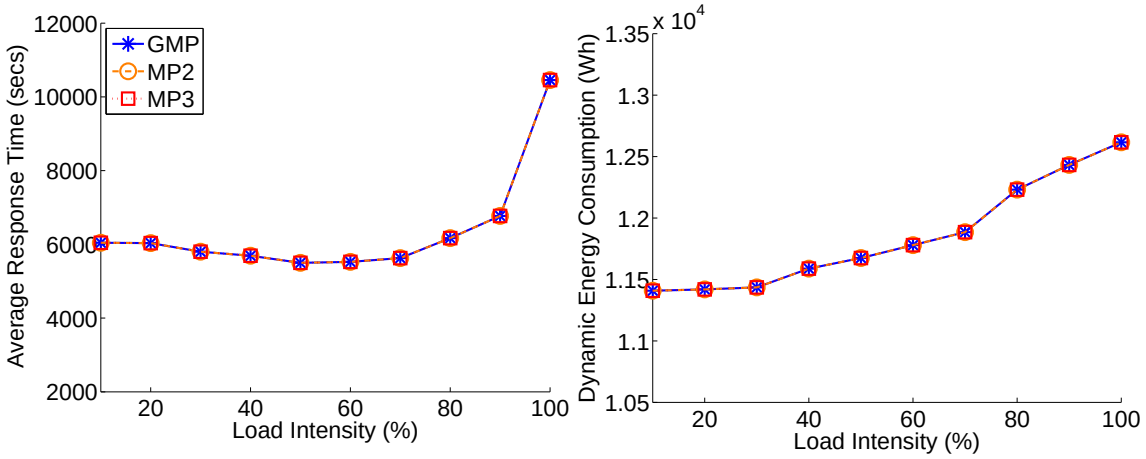


FIGURE 3.15 – Comparaison des temps de réponse moyen et énergie pour les trois heuristiques de placement avec $H_{i,j}^{RE,T} = \langle \overline{H}_{i,j}^{RE}(f), H_{i,j}^{T} \rangle$ pour $f = 0.1$ et $\alpha = 0.4$.

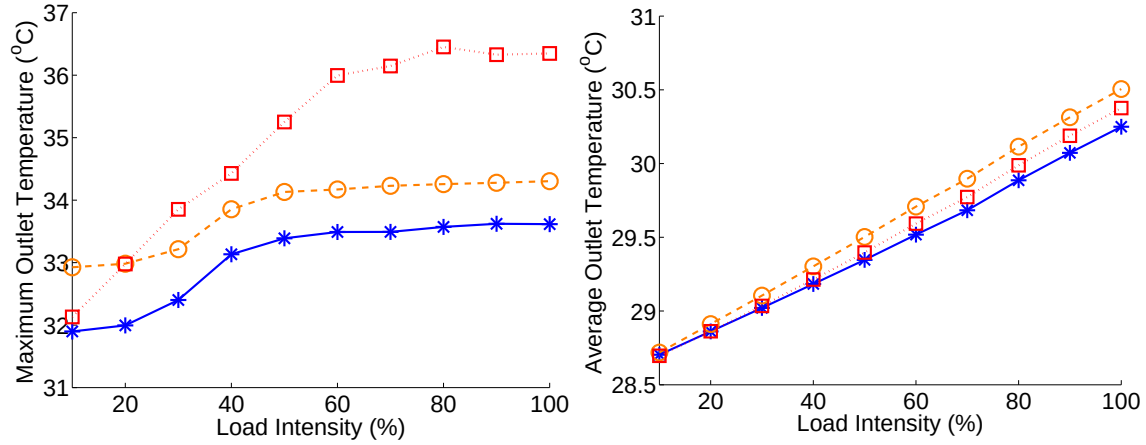


FIGURE 3.16 – Comparaison des températures maximale et moyenne des sorties d'air pour les trois heuristiques de placement avec $H_{i,j}^{RE,T} = \langle \overline{H}_{i,j}^{RE}(f), H_{i,j}^{T} \rangle$ pour $f = 0.1$ et $\alpha = 0.4$.

température des noeuds voisins et sur le coût du refroidissement. Ces résultats confirment que le placement des serveurs affecte l'équilibre thermique ce qui a un impact direct sur l'efficacité et le coût du refroidissement.

3.5 Projets et contextes applicatifs

Ces travaux se sont réalisés dans le cadre de deux projets :

- Le projet Européen CoolEmAll [FCDCO+15] (2011-2014) sur l'efficacité énergétique dans les data-centers. Il regroupait 7 partenaires. Ce projet avait pour but d'optimiser la consommation d'énergie des centres de calculs en optimisant la gestion de la température, des applications, et en concevant un simulateur de centre de calculs (de la gestion de la chaleur jusqu'aux applications). Il a donné lieu a des collaborations internationales notamment avec le laboratoire PSNC en Pologne à travers l'utilisation et l'amélioration du simulateur DCWoRMS.
- le projet national ANR SOP regroupant 5 partenaires dont 3 industriels (2011-2015). Ce projet visait le développement de nouveaux modèles de fonctionnement pour l'informatique des particuliers. Il s'agissait de proposer un modèle de fonctionnement type SAAS/cloud computing et de mettre en place une gestion autonomique et automatique de l'infrastructure afin de ne pas nécessiter d'intervention humaine. Il a donné lieu à la proposition de l'algorithme de placement SOPVP et à son intégration dans OpenNebula. Le modèle hybride proposé par SOP intègre trois paradigmes d'utilisation de ressources : *local* avec les ressources de la machine utilisateur, *en nuage* avec des ressources distantes situées dans des data centers dédiés et *communautaire* en usant des ressources disponibles sur les machines d'autres utilisateurs. Un démonstrateur a été réalisé [SOP].

Dans le cadre des projets CoolEmAll et SOP, j'ai eu des collaborations avec plusieurs industriels membres de ces projets : Christmann (Allemagne), Atos Origin (Espagne), QoS Design (France), Degetel (France), Sysfera (France).

3.6 Bilan

3.6.1 Contributions

Nous avons présenté dans ce chapitre plusieurs approches permettant à travers le placement de machines virtuelles ou physiques de diminuer la consommation d'énergie d'un datacenter. Nous avons proposé une heuristique de réallocation dynamique de VMs utilisant les leviers *live-migration* et *allumage/extinction de machines*. L'heuristique intègre tous les surcoûts induits par ces leviers. Ainsi, la consommation d'énergie provenant des serveurs de calcul est diminuée sans entraîner de dégradation notable sur la QoS. Nous poursuivons en cherchant à améliorer l'approche basée sur la stratégie de gestion des machines *Pivot* avec une approche utilisant les algorithmes génétiques afin d'introduire de l'apprentissage et de la prédiction.

Dans un deuxième temps, nous avons introduit la prise en compte de la chaleur dans les datacenters. Les approches pour contrôler et gérer la température dans les centres de calcul peuvent être classifiées en deux catégories : les approches de gestion dynamique utilisant le DVFS, la migration quand la température dépasse un seuil ou quand un problème thermique est détecté et les approches de placement avec critère de chaleur *thermal aware* visant à utiliser un placement intelligent des tâches sur les processeurs pour éviter des pics de chaleur. Les deux approches sont complémentaires et les solutions que nous proposons dans ce chapitre en sont l'illustration.

Plusieurs méthodes ont été proposées afin de diminuer le coût du refroidissement. Nous avons proposé un modèle temporel et spatial de la température qui tient compte d'aspect statique à travers la matrice de distribution de chaleur et d'aspect dynamique à travers le placement des tâches sur les serveurs.
Le modèle proposé considère un système de refroidissement de type CRAC. Néanmoins, il faut noter qu'il est transposable à d'autres systèmes de refroidissement. Dans le cas des machines frigorifiques air-eau ou eau-eau, le froid est distribué à l'intérieur du centre de calcul par un circuit d'eau glacée sur lequel sont branchés des ventilo-convecteurs. Dans le cas du Free Cooling à air, l'air frais extérieur entre directement dans le centre de calcul ; pour le Free Cooling à eau (pour les centres situés à proximité d'une source d'eau fraîche), l'eau est distribuée dans le circuit d'eau glacée du centre (ou éventuellement encore rafraîchie en amont). Dans tous les cas, le modèle spatial de température peut s'appliquer : la température fournie par l'unité de refroidissement étant soit celle de l'air soit de l'eau. La matrice de distribution de chaleur représente toujours l'impact d'un noeud sur ses voisins.

L'énergie du refroidissement est liée à la température de la sortie d'air. Nous avons proposé d'en diminuer la consommation à agissant à deux niveaux : au niveau serveur en optimisant directement la température de la sortie d'air et au niveau du centre de calcul en modélisant le coût du refroidissement et en le combinant à l'énergie consommée par les serveurs.

Nous avons optimisé le makespan des tâches avec une contrainte de température. En considérant la notion de *thermal load* lors du placement des tâches et en utilisant le levier *DVFS* nous avons ainsi diminué l'énergie. Nous avons proposé une heuristique de résolution *fuzzy based* du problème d'optimisation multi-critères. Cette approche permet d'exprimer des compromis entre critères tout en trouvant des solutions sur la frontière de pareto. Le *fuzzy factor* implémente une préférence utilisateur permettant d'intégrer des priorités entre plusieurs objectifs. Notons que si sa valeur est nulle, le problème devient un placement avec des priorités entre critères.
Sur l'aspect relatif à la prise en compte de la chaleur, il est intéressant de noter également que les ventilateurs eux-mêmes ayant une consommation non négligeable, il peut être intéressant de compléter l'approche en contrôlant leur vitesse à l'aide des commandes ACPI [acp].

Dans ce chapitre c'est donc avec des leviers *middleware*, le placement combiné à des leviers

bas niveau : *migration*, *DVFS*, *on/off* que nous avons économisé de l'énergie. Les informations connues des applications varient des simples besoins en calcul à un profilage de la puissance consommée.

Il faut noter que la consolidation et la prise en compte de la chaleur sont antagonistes : la consolidation entraînera des points chauds. Une solution serait de faire une consolidation avec une contrainte de température. Une autre possibilité serait d'appliquer les heuristiques proposées dans ce chapitre prenant en compte la température en effectuant un placement sur une sous-partie des machines. La consolidation consisterait à prévoir le nombre de machines qu'il faudrait garder allumées et ensuite à appliquer un placement sur ces machines. Pour cela, il sera nécessaire de prédire l'évolution du nombre de machines nécessaires.
Le levier *on/off* est important car il permet d'économiser le coût statique des machines, néanmoins, il a la conséquence de modifier la matrice de distribution de chaleur qui devient dépendante du placement. Une approximation peut consister à la considérer néanmoins constante ce qui revient à calculer le pire cas.

Dans ce chapitre, nous ne proposons pas de comparaison avec l'optimal. Ceci dit, il a été montré dans [Bor13] que les algorithmes de *vector packing* avec deux fonctions de tri permettaient des résultats satisfaisants alors que la résolution de l'optimal avec la programmation linéaire explose avec la taille du problème. De plus, notre approche de fonction de coût composite avec le *fuzzy factor* renvoie des solutions sur la frontière de pareto donc des solutions optimales pour la fonction de coût choisie.

Indépendamment des algorithmes proposés, c'est la démarche scientifique suivie qui est la plus importante. Nous avons proposé des formalisations mathématiques des problèmes de placement, des fonctions objectif et proposé une résolution à base d'heuristiques. L'approche holistique de diminution de l'énergie incluant le refroidissement et les surcouts des leviers verts est importante. Enfin, notre démarche s'est terminée par des expérimentations réelles et simulées permettant de comparer les résultats obtenus par rapport à des algorithmes de la littérature.
Pour les expérimentations en réel, nous avons privilégié une plate forme locale que nous maîtrisions et qui répondait à tous les besoins : un Cloud était opérationnel avec OpenNebula déployé, il était possible d'allumer et éteindre les noeuds, des mesures de consommations étaient disponibles. C'est la plate forme qui nous servait pour tester tous les algorithmes du projet SOP. Toutefois, ces expériences auraient aussi été réalisables sur Grid'5000. Nous avons pu constater les difficultés des expériences réelles : les expériences sont longues, on observe une variabilité liée aux installations dans notre cas d'OpenNebula (temps de réponse d'OpenNebula variables et pouvant conduire à des *timeout*). Tout ceci peut même parfois rendre les expériences non reproductibles. La communauté scientifique s'oriente d'ailleurs désormais plus vers OpenStack.
Les simulations sont en un sens plus simple à réaliser : plus rapides, plus reproductibles, pouvant résoudre des tailles de problèmes plus grandes. Néanmoins, il est important d'in-

tégrer des paramètres réels et il faut noter que généralement les simulateurs ne sont pas immédiats à prendre en main. Le choix du simulateur est important par rapport aux tests souhaités. DCWoRMS est un simulateur sur lequel nous avons maintenant acquis une expertise importante, nous continuerons à l'utiliser dans nos travaux. Nous tenons par ailleurs à souligner que la réactivité et disponibilité des collègues du laboratoire PSNC en Pologne qui ont développé DCWoRMS nous a grandement aidé.

Le point commun de ces travaux est la problématique de gestion de ressources grâce au placement multi-critères d'applications. On notera qu'ici la qualité de service intervient comme critère de placement de manière détournée : nous avons pris en compte la performance dans les critères des problèmes d'optimisations. C'est une problématique que j'avais déjà étudiée en thèse où j'avais proposé une gestion fine des ressources pour des applications parallèles sur grille de calcul afin de garantir une qualité de service prédéfinie. Deux approches étaient proposées : un mode déterministe où tout ce qui s'exécute sur la machine est supposé contrôlé et un mode stochastique où une part de la charge des machines est inconnue [Pas04].

3.6.2 Encadrement et diffusion scientifique

Ces travaux ont été menés en coopération avec deux post-docs : Damien Borgetto (18 mois) et Hongyang Sun (14 mois).

Plusieurs publications ont été réalisées :

- 3 journaux : Journal of Grid Computing, special issue on Green Cloud Computing (à paraître en 2015) [1] ; Ad Hoc Networks Journal (2015) [3] ; Sustainable Computing : Informatics and Systems (Suscom 2014) [5].
- 2 conférences internationales : International Conference on Cloud Networking (CloudNet 2015) [19] ; IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2014) [21].

Chapitre 4

Les reconfigurations autonomiques

4.1 Problématique

Les approches proposées dans les chapitres précédents sont pro-actives : elles visent à anticiper les évolutions du système par détection de phases ou à placer efficacement les tâches sur les serveurs pour les critères que l'on cherche à optimiser. Néanmoins, cela ne nous parait pas suffisant. En effet, les systèmes sont dynamiques et à grande échelle. Des événements peuvent se produire nécessitant des reconfigurations *energy aware*. L'approche proposée dans ce chapitre est une approche réactive basée sur l'informatique autonomique permettant de mettre en oeuvre des politiques de reconfiguration (qui elles peuvent être pro-actives, par exemple en faisant appel à des algorithmes de prédiction ou d'apprentissage) en réponse à différents événements pouvant survenir pendant l'exécution.

Le domaine de recherche de l'informatique autonomique permet de répondre à la complexité grandissante des systèmes informatiques. Paul Horn d'IBM a noté en 2001 le travail nécessaire pour surmonter la complexité de l'informatique [Hor01]. Les systèmes informatiques deviennent de plus en plus complexes nécessitant de la maintenance par des experts informatiques entraînant des délais, des risques d'erreur et des coûts de gestion. Kephart et Chess [KC03] ont défini en 2003 les quatre paradigmes devant être implémentés au minimum par les systèmes informatiques pour qu'ils soient auto-gérés : on parle de propriétés *self-** :

— Auto-guérison (*self-healing*) : cette propriété garantit que le système s'auto-répare suite à des dysfonctionnements de ses éléments. On parle aussi de gestion des pannes qui peuvent être matérielles ou logicielles. Il s'agit de détecter les pannes afin de ramener le système dans un état opérationnel.

— Auto-protection (*self-protection*) : cette propriété garantit que le système ajuste son comportement pour réagir aux attaques de toutes sortes. On parle aussi de gestion de la sécurité que l'on décompose en deux parties : protection interne et protection externe. La protection interne apporte une cohérence globale au système à tout instant (c'est à dire maintient le système dans un état cohérent) et la protection externe contre les événements dit exogènes gère les attaques malicieuses, les infiltrations et autres infections par des virus.
— Auto-configuration (*self-configuration*) : cette propriété permet à un système de s'installer, de se paramétrer et de se mettre à jour tout seul. Il s'agit de gérer de manière autonome le déploiement du système à savoir les tâches suivantes : la configuration, l'installation, le démarrage, la désinstallation, la mise à jour et l'arrêt des éléments d'un système.
— Auto-optimisation (*self-optimisation*) : cette propriété permet au système d'utiliser au mieux au cours du temps les ressources. Cette optimisation répond à différents critères par exemple : performance, qualité de service, consommation d'énergie. Il s'agira par exemple d'adapter le dimensionnement de serveurs en fonction de pics de charge. Ceci permet de s'assurer que le système reste proche d'une performance optimale.

En 2003, Kephart expose dans [KC03] sa vision de l'informatique autonomique. Il s'inspire d'une métaphore biologique du système nerveux humain en le transposant au domaine informatique. En effet, le corps humain s'auto-régule (rythme cardiaque, pression sanguine ...) en fonction des efforts à fournir et des événements extérieurs. Le but de l'informatique autonomique est de surpasser la complexité des systèmes logiciels et matériels en leur procurant des capacités d'auto-gestion.

Pour atteindre les objectifs d'une informatique autonomique, l'idée principale est de s'inspirer des boucles de contrôles utilisées en automatique : un contrôleur guidé par un modèle ajuste en continu des paramètres sur les objets contrôlés en fonction des mesures de retour. IBM a suggéré un modèle de référence pour la boule de contrôle autonomique en informatique [KC03], qui est souvent appelée la boucle MAPE-K (**Monitor**, **Analyse**, **Plan**, **Execute**, **Knowledge**). Ce modèle est de plus en plus utilisé pour détailler les aspects architecturaux des systèmes autonomiques. Il est illustré par la figure 4.1.

Dans la boucle MAPE-K, les **éléments gérés** représentent des ressources logicielles ou matérielles auquel un comportement autonomique est fourni en les couplant avec un **gestionnaire autonomique**. Les **capteurs**, aussi appelés senseurs ou sondes, collectent des informations, aussi appelées métriques, sur les éléments gérés. Les actionneurs prennent en charge les changements à effectuer sur les éléments gérés.

Les problématiques du domaine que nous avons étudiées sont doubles : comment gérer des systèmes informatiques devenus de plus en complexes (c'est à dire comment mettre en oeuvre les propriétés *self-**) ? Comment assurer et maintenir une généricité vis à vis des

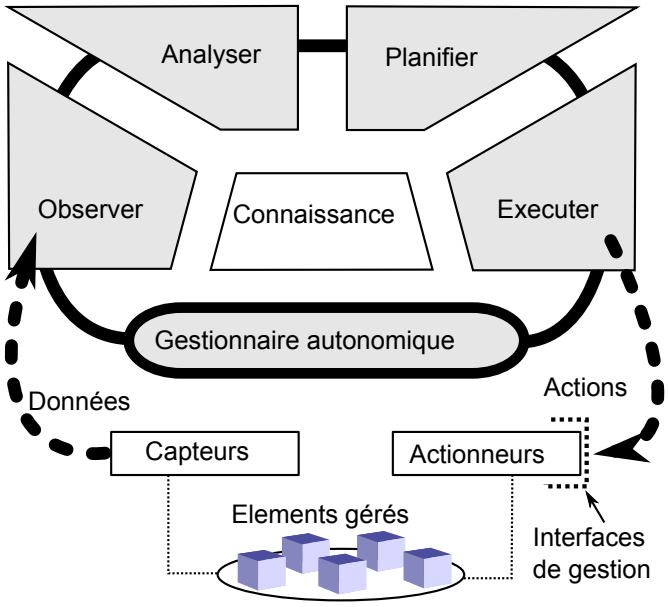


FIGURE 4.1 – Boucle de contrôle autonomique MAPE-K

systèmes existants ?

L'objectif est d'aboutir à une auto-gestion des systèmes avec les propriétés *self-** mais toute la difficulté est de les appliquer aux systèmes informatiques patrimoniaux (propriétaires) déjà existants. Dans les approches proposées, nous viserons des solutions génériques afin de masquer la complexité de bas niveau et ainsi permettre de décrire des politiques de gestion avec un haut niveau d'abstraction.

L'auto-gestion d'un système requiert une modélisation du système pour prendre en compte la correction (les exigences fonctionnelles du système) et la performance (les exigences non fonctionnelles). Cela nécessite d'utiliser des langages de description ou des formalismes appropriés.

Le point commun des travaux présentés dans ce chapitre est de proposer des formalismes génériques pour exprimer les reconfigurations. Les travaux ci-dessous présentent des approches différentes : une basée sur des modèles UML et l'autre basée sur les graphes. Les approches proposées ont été implémentées dans des outils : TUNe (gestionnaire autonomique), AGG *Atrributed Graph Grammar system* et GMTE *Graph Matching and Transformation Engine* (moteurs de graphes). Des expérimentations réelles ont été réalisées en générant des événements entrainant des reconfigurations. L'ensemble des validations pré-

sentées ici utilisent le middleware DIET [1] [CD06] un ordonnanceur de tâches de calculs développé au sein du projet INRIA GRAAL au LIP (ENS de Lyon) et au LIFC (Université de Franche-Comté). Cependant les approches ont été appliquées à d'autres cas d'utilisation. Ces deux approches visent à proposer des reconfigurations dynamiques des systèmes au cours de leur exécution en implémentant les propriétés *self-**.

Dans le domaine des systèmes autonomiques, la notion de correction couvre l'ensemble des états acceptables. Ce dernier est caractérisé par un style architectural (c'est à la caractérisation d'un ensemble de graphes définissant les composants et connecteurs pouvant être utilisés dans ses instances spécifiant ainsi les combinaisons acceptables). Dans l'approche basée sur les graphes, le style architectural est formalisé par les grammaires de graphes. L'auto-protection interne d'un système peut être assurée si toute transformation intervenant dans une politique de reconfiguration est correcte.

Dans le chapitre 2 de ce mémoire, nous avons vu l'application de leviers verts au niveau du système global. Dans ce chapitre, nous exploitons les possibilités du domaine de recherche de l'*autonomic computing* afin de reconfigurer dynamiquement des applications ou des infrastructures conformément à un patron architectural défini. Les reconfigurations de l'infrastructure sont une transposition des propriétés *self-** au matériel. Nous allons dans ce chapitre montrer comment exploiter les reconfigurations autonomiques dans l'objectif d'être plus performant mais aussi en dépensant moins d'énergie. Dans ces études, les connaissances des applications et de l'infrastructure sont beaucoup plus complètes : l'application est notamment connue sous forme de patron d'architecture qui doit être préservé lors des reconfigurations.

4.2 Approches existantes

Plusieurs approches existent dans la littérature pour modéliser les systèmes dynamiques. Les ADL *Architecture Description Language* : langages de description d'architectures proposent une syntaxe et une sémantique rigoureuse pour la description de composants et connecteurs constituants un système, ainsi que pour l'expression de ses propriétés structurelles et comportementales. Les ADLs se concentrent toutefois en général sur des architectures statiques et ne fournissent pas un support satisfaisant pour des aspects dynamiques [HKJHKD05, Gue07, Oqu08]. Parmi les ADLs généralistes classiques permettant la spécification d'évolutions, les plus notables sont Dynamic-ACME [Wil01], Darwin [MK96], Dynamic-Wright [ADG98] et Rapide [LKA+95]. Le π-ADL [Oqu04], un ADL de seconde génération permet une gestion du dynamisme et des transformations horizontales complète : il permet l'addition et la suppression de n'importe quel constituant d'un système.

1. Distributed Interactive Engineering Toolbox

Ces modifications peuvent en particulier être spécifiées en fonction des entrées du système lors de son exécution. Les raffinements et transformations verticales sont spécifiés en π-ARL [Oqu04]. On appelle transformation verticale les changements de granularité ou de précision (on parle aussi de raffinement) ; on appelle transformation horizontale les transformations de modèle et évolutions de système.

Des techniques de représentation reposant sur des modèles génériques haut niveau comme UML fournissent des descriptions formelles ou semi-formelles des propriétés structurelles d'un système [BCDW04], leur côté visuel facilitant une compréhension intuitive de situations complexes. L'adoption de modèles largement répandus en facilite l'usage et la compréhension. UML fournit une définition et une terminologie standardisée pour la description d'un système, facilitant ainsi une compréhension consistante et profonde des architectures logicielles [SX03]. Nous verrons néanmoins dans la suite du chapitre que les descriptions basées sur UML manquent généralement d'outils formels pour garantir efficacement la correction d'un système, dû à l'inhérente semi-formalité d'UML. Après UML, les graphes constituent probablement le modèle le plus largement adopté pour la spécification d'architectures logicielles. Ils bénéficient des mêmes avantages qu'UML couplés à un aspect formel leur permettant d'être au cœur de nombreuses techniques de vérification [RLW13].

Des travaux favorisent l'adoption d'approches hybrides, en liant systématiquement des concepts architecturant décrits par une ADL (ou autre langage de description [LHKJD04]) et représentation d'un modèle générique. Ce faisant, ils proposent des approches hybrides combinant des aspects tirés de ces deux types d'approches. Dans le cadre de la gestion autonomique, on retrouve par exemple des approches [SMS+11, Sha10, BHS+08] basées sur la combinaison de Fractal ADL et d'UML. UML y est utilisé pour la spécification de styles architecturaux, de configurations et de règles de reconfigurations pour des systèmes dont le comportement sous-jacent est géré via une description en Fractal. Le modèle à composants Fractal [BCL+06] permet d'encapsuler les entités logicielles dans des composants et simplifie les actions "du monde réel".

4.3 Approche par modèles UML

Nous allons montrer dans cette approche comment exploiter la modélisation UML afin de définir des politiques de gestion. Dans l'interaction entre le système et son environnement, des sondes détectent des *événements* qui génère des *notifications*. Le gestionnaire autonomique doit ensuite réagir en enclenchant des *actions* sur le système conformément à une reconfiguration exprimée par une politique de gestion de haut niveau. La spécification de politiques de gestion (de reconfiguration) est basée sur la proposition d'un diagramme **PDD** (*Policy Description Diagram*).

4.3.1 Le méta-modèle des diagrammes de reconfiguration : PDD

Ces diagrammes modélisent les politiques sous forme de flot de travail ou "workflow" graphique et sont inspirés des diagrammes d'activité UML [DtH01]. Afin de formaliser de manière précise le modèle introduit, Sharrock a proposé dans [Sha10] une méta-modélisation. La méta-modélisation du niveau graphique a été réalisée à l'aide du formalisme **MOF** "Meta-Object Facility" [Poe06] et la méta-modélisation textuelle a été réalisée en utilisant la forme étendue de Backus-Naur **EBNF** [Sco93]. Nous ne présenterons ici que le méta-modèle sous forme MOF figure 4.2 [SMS+11]. Ce méta-modèle spécifie qu'un PDD est identifié par son nom (attribut *name*) et est composé de plusieurs noeuds et éventuellement de paramètres par les liens de composition entre le stéréotype *PDD* et les stéréotypes *Node* et *Parameters*. Les paramètres sont de type entrants (Paramètre Entrant avec la méta-classe *Input parameter*) ou sortants (Paramètre Sortant avec la méta-classe *Output parameter*) et sont tous ordonnés (à l'aide de l'attribut *order*). Un noeud peut être défini par deux stéréotypes possibles soit d'action (stéréotype *Action*), soit de contrôle (stéréotype *Control*). Le stéréotype d'action décrit les tâches à effectuer pour la politique de gestion modélisée par le PDD. Chaque tâche est décrite suivant une expression (attribut *expression* pour le stéréotype *Action*). Les modifications structurelles (stéréotype *Structural modification*) sont des types d'actions car elles représentent des tâches qui ont un impact direct sur le système géré. Les modifications et sélection de composants (stéréotype *Component modification and selection*) permettent de décrire les modifications d'attributs des éléments du système de manière dynamique pendant l'exécution. Les actions peuvent également donner lieu à des appels de méthode *SWD method call* permettant de décrire des actions sur le système réel (démarrage, arrêt ...). Une action peut aussi référencer un autre PDD (méta-classe *PDD reference*). Le stéréotype de contrôle est quant à lui utilisé pour guider le chemin d'exécution global du PDD, il contrôle la suite logique des tâches pour décrire un flot de travaux de gestion. On retrouve les noeuds de décision, fusion, parallélisation de traitement ou synchronisation.

L'expression de reconfigurations à l'aide de *PDD* a plusieurs avantages : ils sont très simples d'utilisation tout en permettant de décrire de manière détaillée toutes les actions souhaitées sur la réception d'une notification. Ils permettent d'exprimer un mini-algorithme de reconfiguration à partir de variables fournies par des arguments dans les événements provenant des sondes. L'expression des reconfigurations est donc très complète : ajout de composant de l'application, ajout de liaisons, choix décisionnels, introduction de variables, appel de diagrammes avec paramètres d'entrée/sortie ... Nous allons dans la suite de la section montrer des exemples de reconfiguration permettant d'optimiser la gestion des ressources.

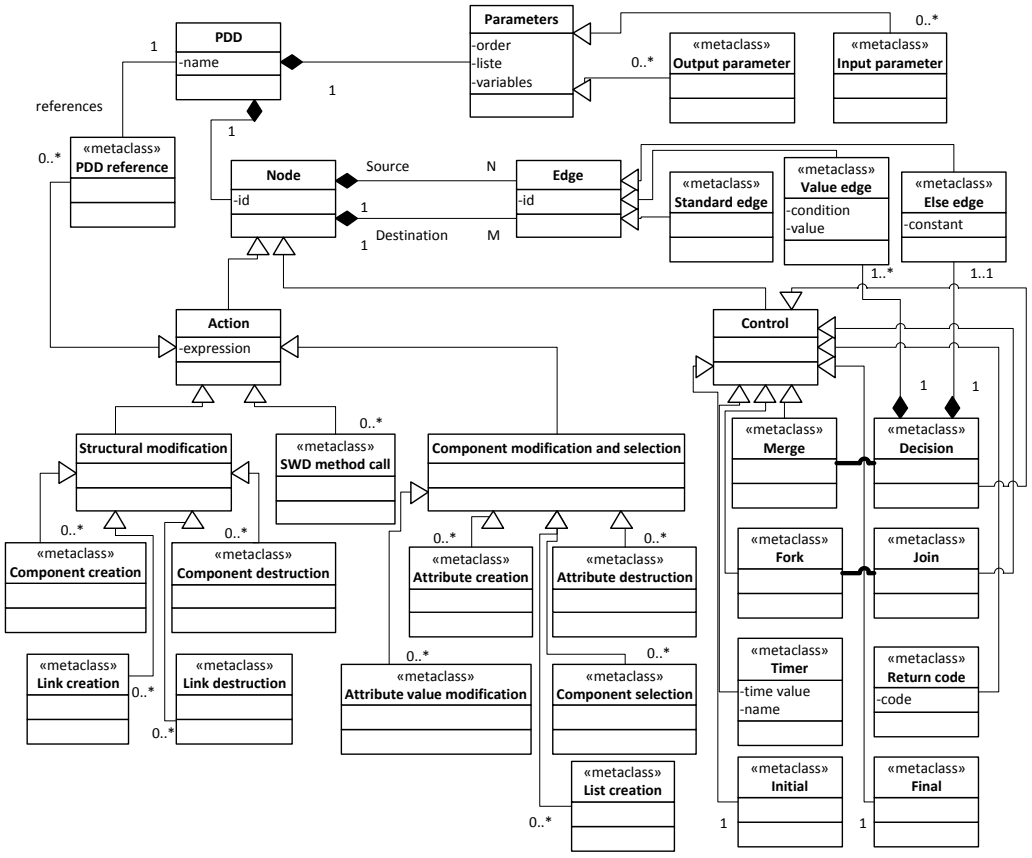


FIGURE 4.2 – Méta-modèle au format MOF des PDD

4.3.2 Reconfigurations dynamiques logicielles : illustration pour du self-healing

Dans allons montré dans cette partie comment les *PDD* permettent simplement d'exprimer des reconfigurations dynamiques logicielles dans un cas d'application de gestion de pannes du système distribué DIET (exemple de mise en oeuvre de la propriété self-healing). Ce cas d'application a été expérimenté sur la plate-forme Grid'5000. DIET [2] [CD06] est constitué

2. Distributed Interactive Engineering Toolbox

d'un ensemble de serveurs de calculs répartis sur des "clusters". DIET permet d'offrir un accès à des services de calculs à des clients répartis sur différents réseaux. La localisation des ressources se fait grâce à plusieurs ordonnanceurs appelés agents, eux-mêmes répartis sur les réseaux des différents "clusters". Les agents sont de trois types et organisés de façon arborescente : MA (Master Agent), LA (Local Agent) et SeD (Server Daemon). Le MA reçoit la requête du client et choisit le serveur SeD pour résoudre le problème. Le LA est un niveau intermédiaire optionnel entre le MA et les SeD. Il est possible d'avoir plusieurs LAs intermédiaires entre un MA et des SeDs. Le déploiement d'une architecture DIET nécessite un service de nommage CORBA (omniNames) appelé par la suite *OMNI*.

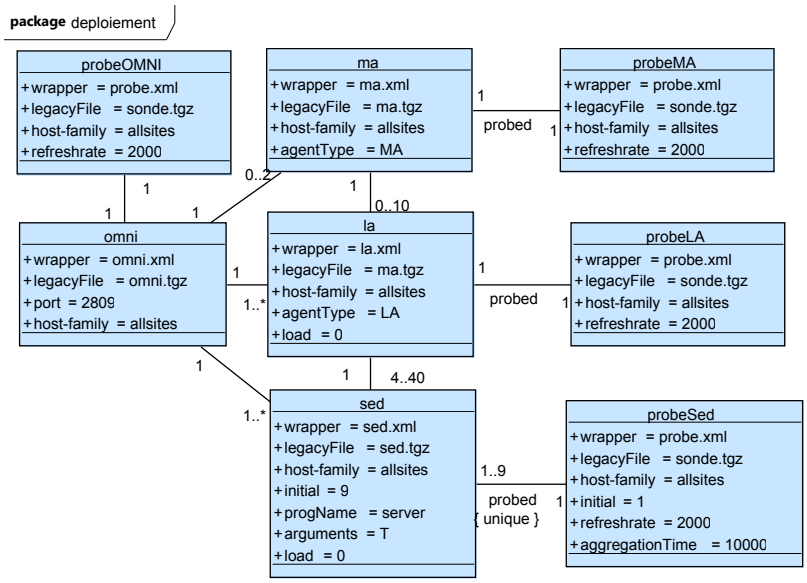


FIGURE 4.3 – Architecture logicielle utilisée pour le déploiement de DIET, 1 sonde d'agrégation pour les SEDs

Dans cette expérimentation nous avons déployé l'architecture présentée figure 4.3 contenant initialement 1 MA, 1 LA, 9 SeDs, leurs sondes respectives et le service de nommage. Nous avons utilisé pour la sonde des SeDs, une sonde capable d'agréger les événements reçus des éléments qu'elle surveille. En effet, lorsque qu'un cluster tombe en panne, tous les éléments du cluster sont potentiellement impactés, ce qui génère plusieurs événements. Dès que la sonde détecte une défaillance, au lieu d'envoyer directement une notification, elle attend pendant un certain temps de détecter d'autres événements. Ce laps de temps est défini par le paramètre *aggregationTime* en millisecondes. Cela permet d'agréger les événements et d'envoyer une liste des éléments défaillants. En contrepartie, la sonde est moins réactive car elle n'envoie la notification qu'après avoir attendu un certain temps.

Dans ce cas d'application de self-healing, nous avons exprimé à l'aide de *PDD*, les politiques

de réparation permettant de restaurer le système dans un état opérationnel. Nous avons déclenché la panne du cluster contenant les SeDs ainsi que le LA les supervisant (les sondes, le service de nommage et le MA étant sur un cluster différent). Nous avons donc exprimé sous forme de *PDD* la reconfiguration du LA et des SeD. Nous ne présentons ici que celui relatif au LA.

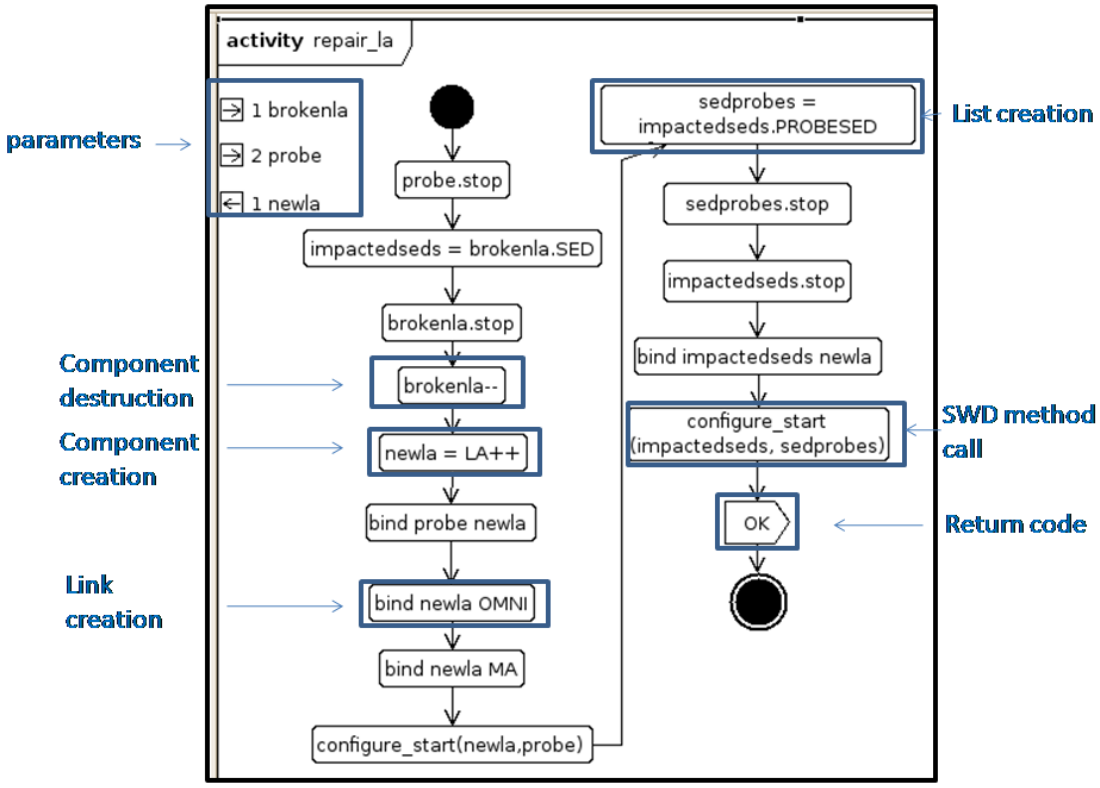


FIGURE 4.4 – PDD utilisé pour la réparation d'un LA

La figure 4.4 montre le PDD utilisé pour la réparation d'un LA. Nous avons mis en évidence les différents éléments du méta-modèle. Ce PDD prend deux paramètres d'entrée : *brokenla* qui contient la liste des LAs en panne et *probe* qui contient la sonde ayant envoyé la notification. Le paramètre de sortie *newla* retourne le LA nouvellement créé lors de l'exécution du PDD. Cette politique de réparation décrit un flot de travail séquentiel d'actions. La première action *probe.stop* (de type appel de méthode) arrête la sonde ayant envoyé la notification afin qu'elle stoppe l'envoi de nouvelles notifications. Une liste *impactedseds* est ensuite créée à l'aide d'une action de type *List creation*. Elle contient tous les SEDs connectés au LA en panne. La politique essaye ensuite d'arrêter le LA en panne

(*brokenla.stop*) et de détruire son composant (*brokenla–*), c'est à dire d'essayer de supprimer tous les fichiers générés et binaires déployés le concernant. Dans le cas d'une panne de cluster, ces deux actions échouent et le flot de travail continue de s'exécuter. Elles peuvent néanmoins être utilisées dans le cas de détection d'un comportement anormal du LA (par des sondes spécialisées). Un nouveau LA est déployé sur une nouvelle machine réservée à l'aide d'une action de création de composants (*newla = LA++*). Ce nouveau LA est ensuite relié à la sonde précédemment utilisée pour le LA en panne (*bind probe newla*) ansi qu'au serveur de nom omniNames (*bind newla OMNI*) et au MA (*bind newla MA*). Une fois tous les liens créés, le nouveau LA et la sonde sont configurés et démarrés à l'aide d'un PDD générique *configure_start* qui fait appel aux méthodes *configure* et *start* de ses éléments passés en paramètre d'entrée. Ce PDD est réutilisé à la fin du diagramme. La partie droite du diagramme s'occupe d'arrêter les SEDs connectés au LA en panne pour les reconfigurer avec le LA nouvellement créé et les redémarrer. Pour cela, les sondes de ces SEDs sont d'abord arrêtés (*sedprobes.stop*) pour ne pas qu'elles envoient de notification lors de l'arrêt de ceux-ci (*impactedseds.stop*). Un lien est ensuite créé avec le nouveau LA (*bind impactedseds newla*). Notons ici que l'ancien lien avec le LA en panne était déjà détruit par la destruction du LA en panne au début de l'exécution du PDD. Enfin, le PDD *configure_start* est utilisé pour configurer et démarrer les SEDs et leurs sondes. Nous voyons ici que pour raison de simplification, les cas d'erreurs ne sont pas traités dans cette politique et le PDD utilise un noeud de type *Return code* pour renvoyer le code de retour *OK* dans tous les cas.

Nous avons mesuré [Sha10] les temps de réparation (environ 8,2 secondes pour le LA sachant que la création du LA et son démarrage prennent 3,5 secondes). Ce temps reste tout à faire raisonnable en comparaison des inconvénients de n'avoir plus toute l'architecture de l'application opérationnelle.

En résumé, les diagrammes *PDD* permettent une formalisation simple à décrire pour l'utilisateur mais puissance en terme de reconfigurations possibles. D'autres noeuds tels que les **fork**, **join**, **timer** ou **codes de retour** permettent de contrôler le flot de travail des actions à mener pour la politique de gestion. Nous l'avons illustré pour un cas de *self-healing* avec DIET mais peuvent être utilisés dans d'autres contextes applicatifs et pour d'autres propriétés.

4.3.3 La gestion de la cohérence : illustration du self-protecting

La gestion de la cohérence (*self-protecting*) est très importante lorsque l'on met en oeuvre des reconfigurations dynamiques autonomes. Nous n'adressons ici que la protection interne du système c'est à dire son maintien dans un état stable et cohérent avec le patron d'architecture spécifié. Nous avons proposé la mise en place de règles de conformité du *SDD Software Description Diagram*. Le *SDD* est le diagramme décrivant le patron d'architec-

ture de l'application, il est basé sur un diagramme de classes UML. Ainsi, par exemple les cardinalités minimales et maximales des classes doivent être respectées lors de toute modification de l'application (ainsi, la création d'un SeD de DIET qui pourrait par exemple être mise en oeuvre pour répondre à un pic de charge de l'application ne peut se faire que si le nombre maximum de SeD que peut gérer le LA n'est pas atteint). Des solutions de récupération de la cohérence ont été introduites et sont effectuées à deux niveaux :

— pendant l'exécution des politiques de gestion *PDD*

— à la fin de l'exécution d'une politique lors de l'arrivée au noeud final d'un *PDD*

Les différentes vérifications sont : le respect des liaisons entre éléments de l'application, le respect des cardinalités minimum et maximum. Pour chaque vérification, un certain nombre d'actions automatisées sont introduites dynamiquement au cours de l'exécution de la politique : création automatique de liaison, création automatique d'instance. Nous limitons toutefois la création automatique d'instance à un seul niveau de création. En effet, il devient difficile de garantir la cohérence du système lors d'une création en chaîne non contrôlée. Ainsi, l'incohérence locale peut persister si cette vérification échoue. Lorsque la vérification de la cardinalité maximum échoue ou que la création automatique d'instance n'a pu avoir lieu on met en place une politique d'interruption. Dans le pire des cas, une procédure d'annulation (rollback) est initiée pour revenir à l'état cohérent initial du *PDD*. Pour cela, il est nécessaire de mémoriser toutes les actions effectuées lors de l'exécution du *PDD*. L'annulation consiste à parcourir toutes les actions dans l'ordre inverse d'exécution. Le tableau 4.5 récapitule les actions et leurs annulations respectives. *++/–* représentent la création/la destruction d'un élément de l'application, *bind/unbind* représentent la mise en place/suppression de liaison entre éléments [SSMG11].

Actions	Annulation
++	- -
- -	++
bind	unbind
unbind	bind

Actions	Annulation
start	stop
stop	start
configure	X

FIGURE 4.5 – Tableaux de transformation des actions en actions d'annulation. A gauche pour les actions de modification structurelle, à droite pour les appels de méthode.

La figure 4.6 représente la vue globale des vérifications et des différents mécanismes mis en oeuvre pour rétablir le système dans un état cohérent.

Noua allons illustrer la gestion de la cohérence proposée avec l'application DIET. Dans l'architecture DIET, les serveurs de calculs SeD peuvent rencontrer d'importantes fluctations de charge liées au nombre de clients à gérer en parallèle. Les LAs doivent répartir les requêtes vers les SeD et peuvent eux aussi voir leur charge évoluer en fonction du nombre de requêtes mais aussi du nombre de SeDs qu'ils ont à gérer. Nous avons introduit une politique de gestion relevant de la propriété *self-optimizing* visant à équilibrer la charge des

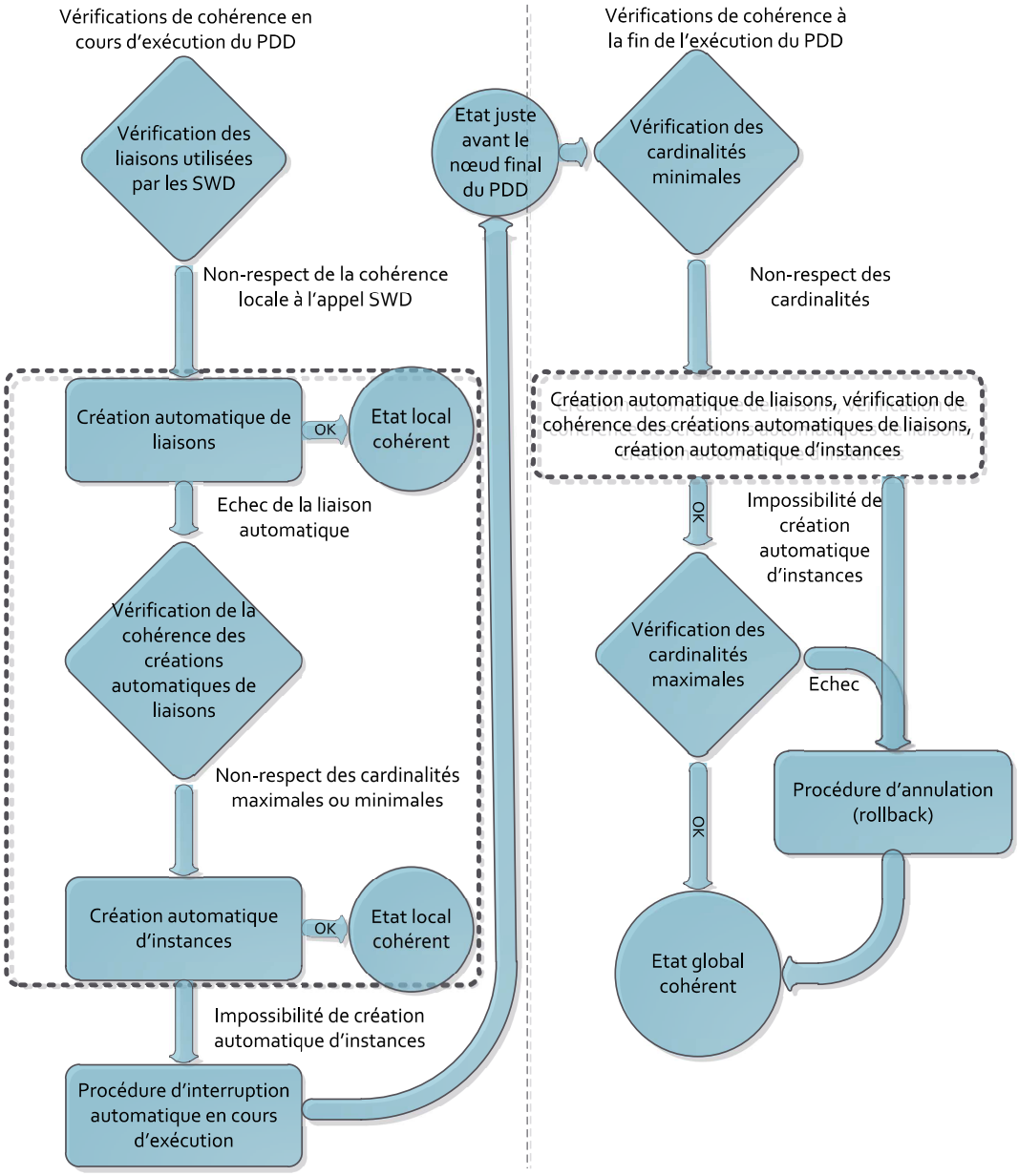


FIGURE 4.6 – Vue globale de la résolution d'incohérences

SeDs ; cette politique globale est illustrée figure 4.7 et est exécutée lors de la réception de la notification *refreshloads*, le premier paramètre d'entrée contient les références des éléments monitorés et leurs valeurs de charge respectives. En fonction de la charge moyenne des SeDs, une politique de création/de destruction de SeDs est enclenchée. La création d'un nouveau SeD illustrée figure 4.8 peut elle-même entraîner la création d'un LA si le nombre maximum de SeD à gérer par un LA a été atteint. Les actions automatiques de la gestion de la cohérence sont indiquées en gris.

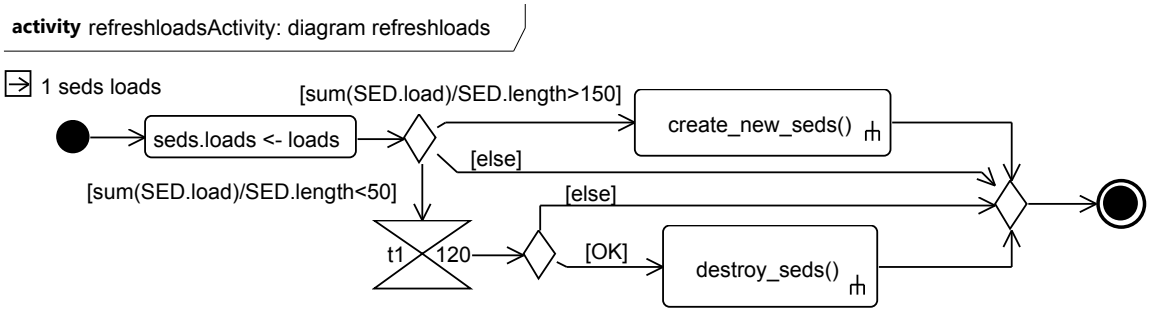


FIGURE 4.7 – PDD global pour l'architecture DIET.

L'objectif du *PDD create new SeD* est de calculer le nombre de nouveaux SEDs à créer pour arriver à une charge moyenne cible stable (pour maintenir la stabilité du système), en utilisant l'équation suivante :

$$x = \left\lceil \frac{\sum_{i=1}^{n} C_i}{T} - n \right\rceil$$

avec x : le nombre de SEDs à créer, C_i : la charge du SED numéro i, n : le nombre actuel de SEDs et T : la charge cible.

Cette équation est exprimée dans la première action du PDD. La charge cible est fixée à 100%, c'est à dire un *uptime* stable et autour de 100%. Les nouvelles instances de SEDs sont connectées au LA approprié. En effet, *list_la = min(LA.load)* crée une liste filtrée des LAs en ne sélectionnant que ceux qui minimisent leur charge CPU. Comme plusieurs LAs peuvent éventuellement avoir la même valeur de charge CPU minimale, au moment de la création de liaison (*bind list_newsed list_la*) ceux qui sont déjà liés à un minimum de SEDs sont choisis. Si tous les LAs sont déjà reliés à leur maximum de SEDs (définit par la cardinalité maximale entre LA et SED), alors d'autres sont créés (l'action de liaison échoue et continue sur le *else*). Dans ce cas de nouveaux LAs sont créés grâce aux appels en référence du PDD *create_new_la*. Le PDD de création du LA est très proche de celui de la création des SEDs et fait appel à la création des MAs en référence (PDD create_new_ma).

Une fois les nouveaux SEDs reliés aux LAs, ils sont tous démarrés avec l'appel de méthode *listenewseds.start*. La méthode de démarrage des SEDs a besoin de parcourir les liaisons

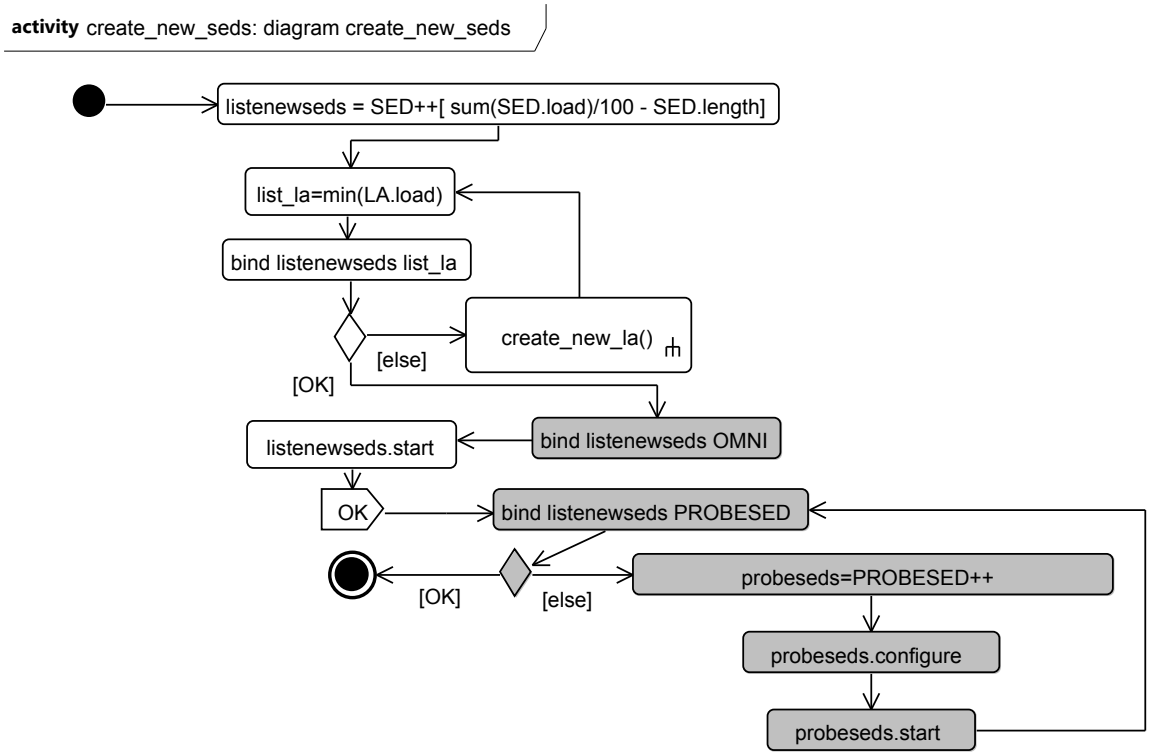


FIGURE 4.8 – PDD avec les actions automatiques en gris pour la création d'un nouveau SED

vers l'OMNI (pour communiquer avec le serveur de nom) et vers le LA (pour se connecter à l'agent Local). Cet appel a donc une incohérence locale puisque la liaison vers l'OMNI n'a pas été créée. Notre approche crée donc une action automatique *bind listenewseds OMNI* placée juste avant l'appel de méthode. Cette action permet de rétablir la cohérence locale pour que l'exécution de l'appel se déroule correctement. Une fois les SEDs démarrés le PDD renvoie le code de retour *OK* et l'exécution arrive au noeud final.

Avant d'exécuter le noeud final du PDD, notre approche fait les vérifications de cohérences finales et éventuellement les procédures de résolution d'incohérences ou d'annulation. Le premier contrôle de cohérence consiste à vérifier si, pour toutes les instances de l'application, les cardinalités sont respectées. Ici, pour les nouveaux SEDs créés, la cardinalité vers les sondes *PROBESED* (1-1) n'est pas respectée puisque la liaison n'a pas été créée. La première procédure de résolution de cette incohérence rajoute l'action de création de liaison *bind listenewseds PROBESED*. Comme chaque sonde PROBESED ne peut être reliée qu'à un seul SED, aucune sonde déjà reliée à un SED n'est disponible pour recevoir une nouvelle liaison. La deuxième procédure de résolution d'incohérence crée donc des instances de PRO-

BESED avec la suite d'actions standard : *probeseds=PROBESED++*, *probeseds.configure*, *probeseds.start*. Cette suite d'actions reboucle sur la création de liaison pour créer autant de sondes que nécessaire. Dans notre cas, il faut exactement le même nombre de sondes que de SEDs nouvellement créés (cardinalité 1-1 dans le patron architectural).

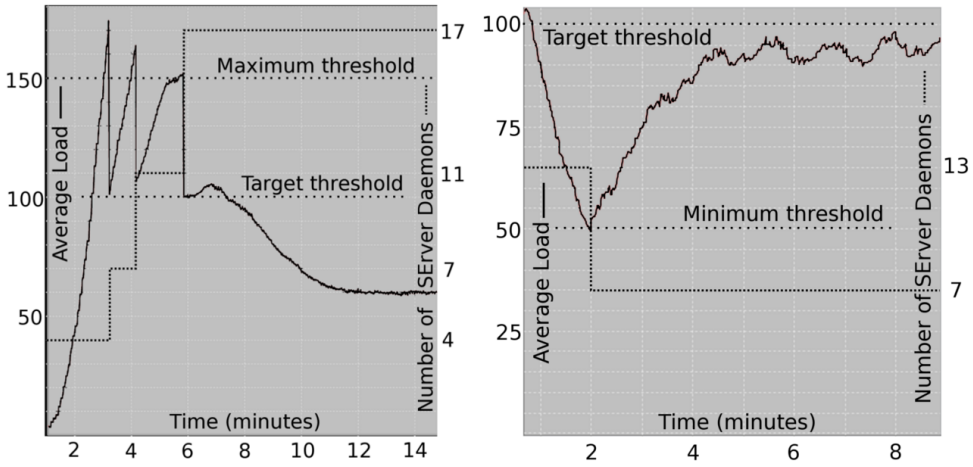


FIGURE 4.9 – Résultat de l'expérience pour (a) gauche : la création de nouvelles instances de SEDs (b) droite : la destruction d'instances de SEDs.

La figure 4.9 (a) montre comment la charge moyenne de tous les SEDs est affectée quand la politique de création des nouvelles instances de SEDs est exécutée. Durant les six premières minutes, la charge moyenne de tous les SEDs est en train d'augmenter rapidement et dépasse le seuil maximum (maximum threshold) trois fois. Un nombre calculé de nouveaux SEDs sont créés pour atteindre la charge optimale CPU de 100%. Trois nouveaux SEDs sont créés la première fois, quatre la deuxième fois et six la troisième fois. Nous pouvons voir que la charge moyenne continue d'augmenter un petit peu au dessus du seuil maximum à chaque fois. Ceci est dû au fait qu'il faut quelques secondes pour créer et démarrer tous les nouveaux SEDs. De t=7 à t=15 minutes, 17 serveurs sont en marche et absorbent les requêtes des clients. Leur charge moyenne se stabilise autour de 60% à t=10 minutes. Le petit dépassement du seuil cible à 6 minutes est dû au fait que l'ordonnancement interne de l'architecture DIET est par défaut un *round robin* qui ne donne pas la priorité à l'utilisation des SEDs nouvellement créés. De ce fait, quelques requêtes clients sont toujours envoyées aux SEDs surchargés jusqu'à ce que l'ordonnancement atteigne les nouveaux.

La figure 4.9 (b) montre un exemple d'exécution de la politique de destruction des SEDs sous-chargés *destroy_seds*. Pour cela, nous avons d'abord surchargé 13 SEDs puis diminué le nombre de requêtes à une requête toutes les secondes, avec un calcul demandant six secondes CPU pour que leur charge moyenne passe en dessous du seuil minimum (*minimum*

threshold). Le nombre de SEDs à supprimer est calculé de la même manière que pour la politique de gestion des SEDs surchargés. Quand un SED est supprimé, il ne reçoit plus de requêtes et termine les calculs en cours. Une fois toutes les requêtes terminées, le gestionnaire autonomique arrête tous les processus, nettoie les fichiers de travail et rend les machines disponibles à nouveau. Nous voyons que la charge moyenne se stabilise autour de 95% de t=2 à t=4 minutes. En effet, les requêtes sont redirigées vers les SEDs restants qui ne sont pas supprimés et ce temps de transfert de charge et de stabilisation peut éventuellement prendre plus de temps. Il dépend du taux de requêtes et du temps de calcul de chaque requête. C'est la raison pour laquelle le timer t1 est utilisé. En effet, le timer t1 attendrait deux minutes pour la prochaine exécution du PDD de destruction des instances de SEDs si la charge restait sous le seuil minimum (ce qui n'est pas le cas ici).

Cette expérimentation a été exposée dans [SSMG11]. Les *PDD* permettent de spécifier des politiques de gestion variées permettant de répondre à différents objectifs d'optimisation. Les propriétés *self-** peuvent être appliquées aux applications sans les modifier. Nous l'avons illustré ici avec l'application DIET mais d'autres cas d'application ont été présentées dans [Sha10]. Nous avons donc garanti la généricité ainsi que la cohérence globale lors de mise en oeuvre des politiques.

Toute cette approche a été initiée suite à la thèse de Laurent Broto [Bro08] encadré par Daniel Hagimont. L'ensemble de ces contributions a été implémenté dans l'outil TUNe (Toulouse University Network). TUNe [BSB+08] [BHS+08] [Bro08] est un gestionnaire autonomique basé sur un modèle à composants. Sa particularité est de permettre d'ajouter des comportements autonomiques à différents types de logiciels patrimoniaux déjà existants. Il offre une vision uniforme par l'encapsulation des logiciels administrés dans des composants.

4.4 Approche basée sur les graphes

La gestion de la cohérence présentée dans la section précédente a toutefois des limites : un rollback est nécessaire en cas de problème nécessitant une mémorisation des actions mises en oeuvre.

D'autre part, dans la formalisation du style architectural de l'application (diagramme de classes UML), il n'est pas possible d'exprimer des relations récursives.

Nous allons montré comment une approche basée sur les graphes développée dans le cadre des travaux de thèse de Cédric Eichler [Eic15] peut gérer le problème de la cohérence interne lors de reconfigurations dynamiques.

L'intérêt d'une approche basée sur les graphes pour gérer des reconfigurations dynamiques est double : la formalisation est souple et intuitive et surtout la cohérence interne sera gérée

de manière plus simple. On parlera de reconfigurations correctes par construction (*correct by design*).

Les graphes permettent de représenter de manière intuitive des situations complexes dans lesquels les noeuds représentent les éléments du système (composants d'une architecture logicielle, machines pour la description d'architectures physiques ...) et les arcs représentent les interactions, connexion .... A un instant t, un graphe permet de représenter la configuration actuelle d'un système. Afin de représenter un système dynamique, il est nécessaire de spécifier ses évolutions (c'est à dire les états acceptables dans lesquels il peut se trouver) et qui résultent de reconfigurations. Pour cela, on peut définir un style architectural. La réécriture de graphes permet de spécifier les transformations (reconfigurations) applicables sur un graphe donné. Les transformations de graphe modélisent les systèmes dynamiques par des graphes et sont basées sur des relations entre graphes (plus précisément à la recherche de motifs (morphisme) dans un graphe). On trouve dans la littérature deux approches permettant de définir des règles de réécriture et basées sur la théorie des catégories : SPO *Single PushOut* [Awo06] et DPO *Double PushOut* [CMR+97].

Les grammaires de graphe sont basées sur les grammaires génératives et forment un formalisme expressif pour la spécification de logiciels et de styles architecturaux. Elles définissent un ensemble de graphes qui sont des configurations acceptables du système. La caractéristique générative d'une grammaire permet d'exprimer des reconfigurations (les règles de production de la grammaire) correctes par construction. C'est donc un formalisme intéressant pour exprimer la propriété de *self-protecting* interne des systèmes autonomiques. En effet, cela garantit qu'au cours des modifications dynamiques le système restera cohérent.

Nous allons dans les sections ci-dessous reprendre les différents éléments d'une reconfiguration dynamique basée sur les graphes : la modélisation d'un système à l'aide d'un style architectural, l'expression de règle de transformation et l'évaluation d'une reconfiguration.

4.4.1 Modélisation d'un système : le style architectural

Les composants du système, leurs relations et leurs propriétés sont représentés grâce à un graphe dirigé et partiellement attribué (c'est à dire que tous les éléments ne sont pas nécessairement attribués).

Les grammaires de graphes sont adoptées en tant que méta-modèle pour la formalisation de styles architecturaux. Une configuration est considérée correcte vis à vis d'un style si elle en est une instance. Une transformation est à son tour considérée correcte vis à vis d'un style si elle le préserve, c'est à dire que son application à une configuration correcte produit nécessairement une configuration correcte.

Pour la modélisation basée sur des graphes on trouve dans la littérature principalement deux méthodes pour la caractérisation de style architectural : les graphes types [BHTV06], [WF02] et les grammaires de graphes [LM96], [HIM99], [HM00]. Les graphes types [CMR96] sont similaires aux diagrammes de classes UML. Les graphes types sont faciles à utiliser et à interpréter, en particulier grâce à leur similitude avec UML, un modèle familier et répandu. Ils sont toutefois moins expressifs que les grammaires de graphes, qui permettent par exemple de spécifier simplement des relations de précédence. Nous adoptons, pour la représentation de style, ce second formalisme. Cette décision est motivée en particulier par l'une des particularités les plus notables des grammaires de graphes : leur aspect génératif.

Les grammaires de graphes ont pour origine les grammaires génératives de Chomsky. De telles grammaires sont en général spécifiées par un axiome, des termes terminaux et non-terminaux et un ensemble de règles de réécriture (de chaînes de caractères) appelées productions de la grammaire. Grâce à la réécriture de graphe, ces grammaires peuvent être transposées aux graphes. Une grammaire de graphe est définie par un système (AX, NT, T, P) où

1. AX est l'axiome. Il peut être représenté soit par un nœud fictif soit par un graphe symbolisant le motif minimal de l'application.
2. NT est l'ensemble des types de nœuds non terminaux.
3. T est l'ensemble des types de nœuds terminaux.
4. P est un ensemble de règles de réécriture de graphes appelées productions de la grammaire. Ces règles sont constituées des termes de la grammaire, de sorte que tout nœud d'une production est unifiable avec au moins un terme de T ou NT.

Une des particularités les plus notables de ces grammaires est leur aspect génératif. Les règles de production de la grammaire de graphe formalisent la construction de ses instances en définissant quand et comment un composant peut être déployé. Un graphe est une instance d'une grammaire (AX, NT, T, P) pouvant être obtenu à partir de l'axiome AX en appliquant une suite de règles de production de P.
Si l'instance ne contient pas d'élément de NT, elle est dite **consistante**. Les règles de production peuvent être considérées comme *correctes par définition*, l'application d'une production p à un état acceptable (i.e., un état pouvant être atteint à partir de l'axiome en lui appliquant une suite (p_i) de règles de productions) résulte nécessairement en un état acceptable (c'est à dire un état pouvant être atteint à partir de l'axiome en lui appliquant la suite de règles de production $(p_i) + p$).

Les propriétés des éléments d'un système sont représentées par des attributs. Il y a différentes méthodes : soit intégrer les attributs comme des noeuds particuliers du graphe [EEPT06], [Ren] ce qui a comme conséquence d'augmenter la taille du graphe et donc complexifie la recherche de morphisme ; soit solution plus simple, affecter aux noeuds et aux arcs une liste de couples représentant des attributs et leurs domaines de définition

[Gue07], [RDC+10], [Tae]. Dans l'approche que nous proposons nous utilisons la seconde méthode en étendant ce que l'on peut trouver dans la littérature pour la représentation des attributs variables.

Nous illustrons la grammaire de graphes en reprenant l'application DIET. Dans les travaux présentés dans la section 4.3, DIET est décrite en utilisant des diagrammes de classes. La gestion de la correction s'effectue par la vérification *a posteriori* de contraintes de style à la fin de l'exécution d'un *PDD*. Ainsi, des roll-backs peuvent s'avérer nécessaires lorsque ces dernières ne sont pas satisfaites. De plus, il n'était pas possible d'exprimer le fait qu'un LA puisse en gérer un autre ; les configurations de DIET intégrant plusieurs niveaux de LAs ne pouvaient pas être exprimées. Dans la grammaire de graphes que nous proposons pour DIET, nous considérons les attributs suivants :

- chaque composant a un identifiant unique (de l'ensemble ID)
- chaque composant correspond à un type de l'ensemble Nat pouvant prendre les valeurs $\{"OMNI", "MA", "LA", "SED"\}$
- pour chaque SED, nous caractérisons l'ensemble des services qu'il peut fournir par un attribut qui est un élément de $Serv$

GG_{DIET}, la grammaire de graphe spécifiant formellement DIET est définie par $GG_{DIET} = (AX_{DIET}, NT_{DIET}, T_{DIET}, P_{DIET})$ avec

- les noeuds non terminaux définis par $NT_{DIET} = \{v_{temp}(("Temp", \{"Temp"\}))\}$,
- les noeuds terminaux définis par $T_{DIET} = \{T_o, T_a, T_{sed}\}$,
- les règles de production définies par $P_{DIET} = \bigcup_{i\in[1,4]} p_i$. Les règles de production de la grammaire de graphe formalisent la construction de ses instances en définissant quand et comment un composant peut être déployé.

Avant d'exprimer les règles de production de la grammaire de DIET, il est nécessaire d'expliquer le formalisme que nous avons utilisé pour exprimer les règles de réécriture de graphe. L'approche la plus simple pour transformer un graphe G en un graphe G' est de remplacer un sous-graphe L de G, en un graphe fils R. Dans ce cas, une règle de réécriture est décrite par la paire de graphes (L,R). Son application à un graphe G est contrainte par l'existence d'au moins une occurrence de L dans G et a pour conséquence la suppression d'une occurrence de L du graphe G et son remplacement par une copie (isomorphe) de R. Cette suppression peut toutefois entraîner l'apparition d'arcs sans nœud de départ ou d'arrivée ou sans aucun des deux. On parle dans ce cas d'arcs suspendus. Deux approches dans la littérature résolvent cette problématique : l'approche SPO [Lö93], [EHK+97] et l'approche DPO [CMR+97]. Dans approche SPO, les arcs suspendus sont supprimés. Dans l'approche DPO, un graphe interface K est explicitement spécifié. Une règle est alors notée $L \leftarrow K \rightarrow R$. Le graphe K (Ker ou Kernel) spécifie la partie invariante vis à vis de l'application de la règle. Dans l'approche DPO, une règle ne peut pas être appliquée si son exécution entraîne l'apparition de tels arcs. Cette condition supplémentaire est appelée

condition de suspension.

Dans l'approche que nous proposons, nous exprimons les règles de réécriture selon l'approche SPO. En effet, elle est plus expressive que DPO et est le plus utilisée. Nous avons transposé cette approche d'un point de vue catégoriel vers un point de vue ensembliste avec des graphes avec mono-arcs (un seul arc entre deux noeuds). Les arcs suspendus sont effacés comme dans l'approche SPO classique.
Dans le cas de la grammaire de DIET, nous devons définir 4 règles de production. La première règle (p_1) définit l'initialisation consommant le nœud axiomatique AX. Le service de nommage et le MA sont alors déployés, ainsi qu'un nœud temporaire garantissant que le MA gère au moins un composant. Ce nœud sera instancié plus tard en un LA ou un SED. Finalement, le MA s'enregistre au service de nommage. Cette règle $p_1 = (L_{p_1} \xrightarrow{m_{p_1}} R_{p_1})$ est illustrée dans la figure 4.10.

La règle de production p_2, représentée dans la figure 4.11, modélise l'ajout d'un nœud temporaire géré par le MA ou un LA. Ce nœud temporaire a pour vocation à être instancié en un LA ou un SED. Son morphisme associé, m_{p_2} est caractérisé par la fonction associant $v_1 \in V_{L_{p_2}}$ à $v_1 \in V_{R_{p_2}}$. Cette règle permet d'exprimer l'ajout d'un fils à un MA ou LA pouvant donc être soit un LA soit un SED.

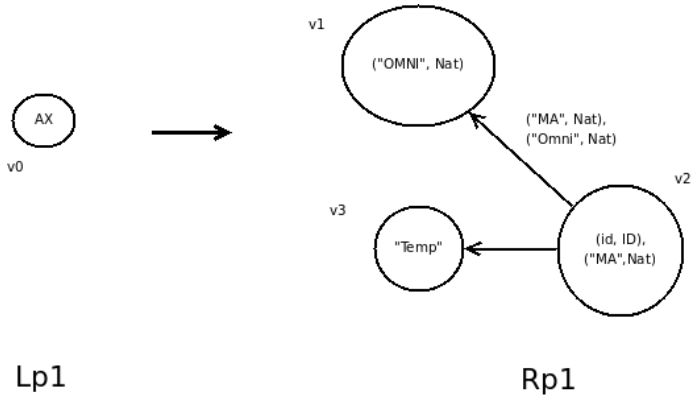


FIGURE 4.10 – Initialisation (p_1)

L'instanciation d'un nœud temporaire en un SED ou un LA est décrite par les règles de production p_3 et p_4, introduites dans les figures 4.12 et 4.13, respectivement. Lors de l'instanciation d'un noeud non terminal en un LA, le noeud v3 est supprimé lors de l'application de la règle : il est dans L mais pas dans R. Les noeuds v1 et v2 sont invariants (à la fois dans L et R), les noeuds v4 et v5 sont ajoutés (présents uniquement dans R). Le déploiement d'un SED ou d'un LA implique son enregistrement au service de nommage. Comme un LA doit gérer au moins un composant, un nœud temporaire est ajouté lors de l'instanciation correspondante.

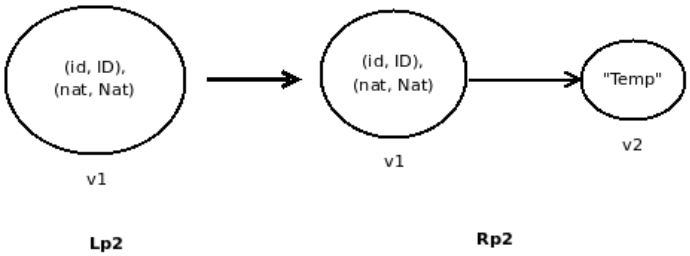


FIGURE 4.11 – Ajout d'un nœud temporaire (p_2)

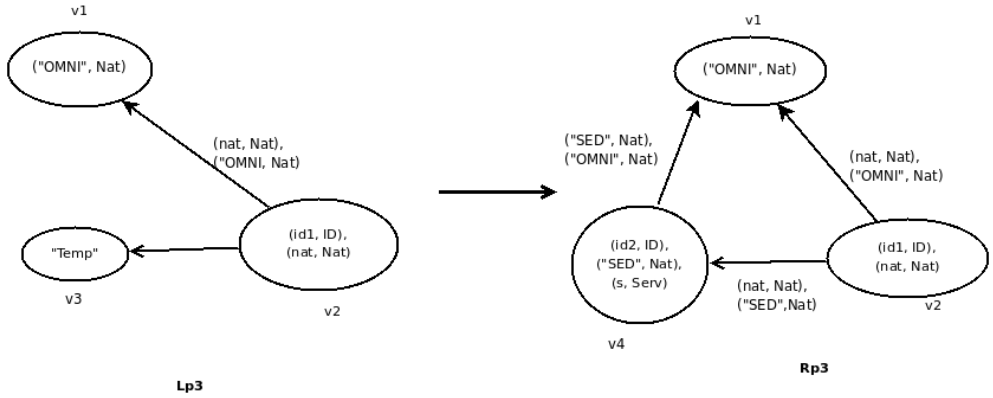


FIGURE 4.12 – Instanciation d'un nœud non terminal en un SED (p_3)

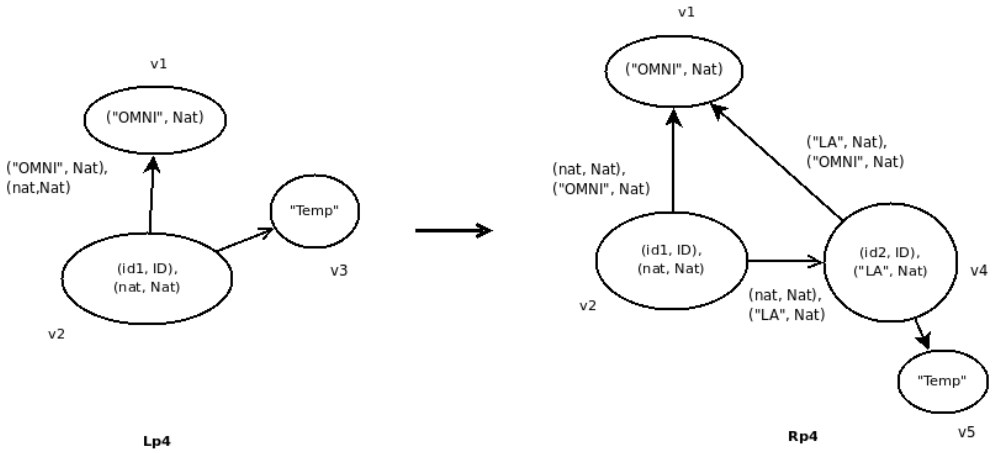


FIGURE 4.13 – Instanciation d'un nœud non terminal en un LA (p_4)

La configuration finale, notée G_6 sur la figure 4.14, contient un serveur de nommage et un

MA gérant un LA. Ce dernier gère à son tour un SED et un LA sous lequel se trouve un deuxième SED. G_6 peut être obtenue en appliquant successivement à l'axiome la suite de règles de production $(r_i)_{i\in[1,6]}$ où $r_1 = p_1$, $r_2 = r_5 = p_4$, $r_3 = r_6 = p_3$ et $r_3 = p_2$.

Notons G_i chacune des instances intermédiaires. On peut également écrire $AX \xrightarrow{(p_1,h_1)} G_1 \xrightarrow{(p_4,h_2)} G_2 \xrightarrow{(p_3,h_3)} G_3 \xrightarrow{(p_2,h_4)} G_4 \xrightarrow{(p_4,h_5)} G_5 \xrightarrow{(p_3,h_6)} G_6$. Finalement, bien que chaque G_i soit une instance de la grammaire, et donc une configuration correcte de DIET, G_3 et G_6 sont les seules à ne pas contenir de terme non terminal (aucun noeud du graphique contenant le label "Temp"), et donc à être des instances consistantes.

Le tableau 4.1 récapitule les concepts liés aux systèmes dynamiques et la formalisation associée dans le cadre de l'approche que nous proposons.

Concept	**Formalisation**
Composant/Constituant élémentaire	Nœud attribué
Connexion/Relation	Arc attribué
Propriété	Attribut
Configuration/État du système à un instant donné	Graphe partiellement attribué
Transformation/Évolution du système	Règle de réécriture de graphe

TABLE 4.1 – Récapitulatif de la formalisation des différents concepts liés aux systèmes dynamiques

Dans la section suivante nous allons détailler comment exprimer les reconfigurations à l'aide des règles de réécriture de graphes.

4.4.2 Les règles de transformation : l'expression de reconfiguration

Par conséquent, la spécification d'un style à l'aide d'une grammaire fournit un ensemble de règles correctes par définition. Adopter cette représentation fournit donc une base de travail des plus intéressantes pour l'étude de la conservation de la consistance dans des systèmes adaptables. Les règles de réécriture de graphe spécifient les transformations et les conditions d'application. Les évolutions d'un système sont modélisées par des transformations de graphes symbolisées par des règles de réécriture de graphes. Une transformation (donc une règle de réécriture) est dite correcte vis à vis d'une grammaire si l'application d'une règle à une instance de la grammaire produit une autre instance de la grammaire.

Les travaux de Cédric Eichler [Eic15] adoptent les bases du modèle proposé par [Gue07] et propose un formalisme sur lequel les règles de réécriture se basent : l'expansion et la restriction. L'expansion correspond à la fusion de deux graphes (c'est à dire l'union) ; la restriction correspond à l'intersection de deux graphes. Le formalisme proposé intègre une

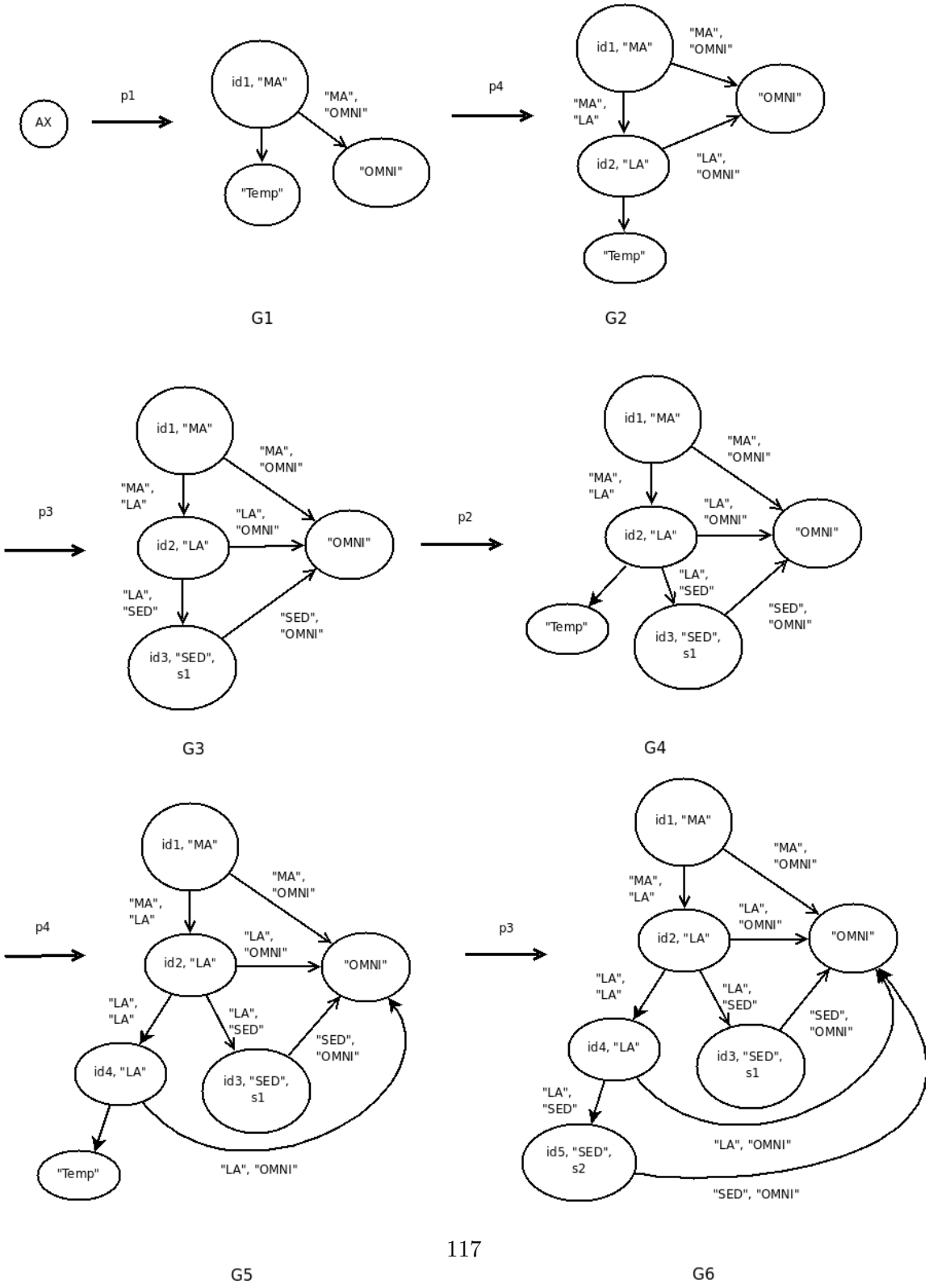


FIGURE 4.14 – Génération d'une configuration correcte de DIET

extension pour l'intégration d'attributs variables et une prise en compte de l'attribution partielle.

Toute grammaire générative est correcte par définition puisque les règles de production garantissent la correction. Dans ses travaux, Cédric Eichler a prouvé une correction par construction de trois opérations sur les règles d'écriture :

— Inversion d'une règle : l'inversion exploite la propriété d'inversibilité des règles de réécriture [Roz97]. Elle consiste à définir une transformation inverse annulant les effets d'une autre. L'opération d'inversion consiste à créer, à partir d'une règle de réécriture, une règle inverse. Trois conditions sont nécessaires pour que l'inversion conserve la correction d'une règle correcte r.
Tout d'abord, l'application de la règle correcte r ne doit pas pouvoir faire apparaître d'arc suspendu.
Deuxièmement, la règle inverse ne doit être applicable que si elle annule (théoriquement) l'effet d'une précédente application de r.
Troisièmement, une transformation inverse t^{-1} annulant l'effet d'une transformation t, ne doit pas invalider les conditions d'applicabilité des transformations ayant eu lieu entre t et t^{-1}. En particulier, t^{-1} ne doit supprimer aucun élément nécessaire à l'application d'une telle transformation intermédiaire. Dans la réécriture de graphe, cette notion de nécessité se traduit par la présence d'un élément dans la partie gauche de la règle entrant dans la spécification de la transformation. Il s'agit donc d'implémenter un compteur de reconnaissance gardant une trace, pour chaque élément d'une configuration, du nombre de fois où il a fait partie du motif permettant l'application d'une transformation. Un élément peut ainsi être effacé si un tel compteur est nul. Cédric Eichler a proposé un algorithme dans sa thèse permettant d'implémenter ce mécanisme et ainsi d'augmenter automatiquement une grammaire de graphe sans modifier le style architectural qu'elle caractérise. Ainsi, par l'intermédiaire de la gestion d'un compteur, il a montré que l'inversion d'une règle conservait sa correction.

— Composition de règles : il s'agit du processus créant une règle de réécriture composée à l'aide de deux autres règles. La composition de règles correctes conserve la correction.

— Spécialisation d'une règle : la spécialisation d'une règle (c'est à dire un cas réduit d'applicabilité) conserve la correction des transformations.

Utiliser une grammaire de graphes comme style architectural pour les systèmes autonomiques est particulièrement intéressant puisqu'il devient possible de définir des règles de transformations correctes par construction et permet donc de garantir le maintien du système dans un état correct lors des transformations dynamiques. La propriété *self-protecting*

interne est donc garantie et vérifiée lors de la phase de conception du système. Une transformation correcte ne pouvant pas par définition produire un état incorrect, aucune annulation (roll-back) suite à une transformation n'est donc nécessaire.

Le tableau 4.2 récapitule les notions appliquées aux systèmes autonomiques et provenant de la théorie des graphes.

Concept	**Formalisation**	**Signification**
Style architectural	Grammaire de graphe	Caractérisation d'un ensemble de graphes
Correction : Configuration	Instance de la grammaire	Instance du style architectural
Transformation	Transformation pour laquelle l'ensemble des instances de la grammaire est stable	Transformation préservant la correction du système

TABLE 4.2 – Récapitulatif : style architectural et correction

La propriété de *self-protecting* est vérifiée grâce à l'approche basée sur les graphes de manière intrinsèque et permet également de vérifier les autres propriétés *self-** comme l'a illustré Cédric Eichler dans sa thèse [Eic15].

4.4.3 L'évaluation d'une reconfiguration

L'évaluation d'une reconfiguration est nécessaire et est la base de la propriété de *self-optimizing*. Les propriétés d'un système sont formalisées par des attributs du graphe et la qualité d'une reconfiguration au regard de critères fonctionnels ou non fonctionnels est reflétée par des contraintes.

Un attribut est un couple $Att = (A, D)$ où A est sa valeur pouvant être une constante, une variable ou une expression et D est son domaine de définition. Les contraintes sont introduites dans la formalisation comme des attributs particuliers. Une contrainte *Cons* est un attribut (A_{Cons}, D_{Cons}) où D_{Cons} = {"vrai", "faux", "inconnu"}.

Nous définissons un AC-graphe comme étant un graphe dirigé partiellement attribué accompagné d'un ensemble de contraintes : un AC-graphe G est défini comme un système (V, E, Att, $Cons$) où :

1. V et $E \subseteq V^2$ correspondent respectivement aux ensembles des nœuds et arcs du graphe, de sorte que (v_1, v_2) désigne un arc de v_1 vers v_2.
2. Att et $Cons$ sont des familles d'attributs et de contraintes indexées par deux sous-ensembles de $V \cup E \cup \{G\}$.

Dans l'approche présentée, nous proposons d'intégrer les attributs et contraintes de la même manière, ce qui permet de gérer des attributs inter-dépendants et d'intégrer des contraintes lors des reconfigurations. Les approches classiques de réécriture de graphe permettent à une règle de modifier uniquement les attributs y apparaissant. Toutefois, une limitation apparait en présence d'inter-dépendances, une telle modification doit être propagée aux attributs dépendants. Ceci peut entraîner un grand nombre d'applications de règle(s). Par exemple, lors du déploiement d'un SED sous un LA, l'ensemble des services fournis par ce dernier doit être mis à jour en conséquence. En fait, cette mise à jour doit être récursivement impactée sur chaque ancêtre du LA, jusqu'à atteindre le MA ou un LA dont l'ensemble de services reste inchangé (car il fournissait déjà tout service proposé par le nouveau SED). Dans le scénario précédemment décrit, et puisqu'une modification ne peut être menée à bien que sur un attribut apparaissant explicitement dans une règle, il y a autant d'applications que de LAs modifiés. Ce phénomène "domino" implique une grande perte d'efficacité et d'élasticité. Il a donc été nécessaire d'enrichir les systèmes de réécriture afin de gérer efficacement les attributs dynamiques et interdépendants.

Dans les grammaires de chaînes de caractères classiques, ce type de modification peut être effectuée grâce à l'adjonction, dans les règles de réécritures, d'algorithmes permettant la mise à jour d'attributs appelés mutateurs [PF11]. Un mutateur est un algorithme qualifié d'interne s'il met à jour des valeurs d'attributs ou de contraintes et d'externe s'il applique une méthode.

Nous avons donc proposé dans nos travaux une transposition des mutateurs qui existent dans les grammaires de chaines de caractères classiques aux graphes : ainsi, les évolutions d'attributs lors de reconfigurations sont formalisées grâce aux mutateurs. Nous verrons un peu plus loin dans le chapitre que notre solution à base de mutateurs s'avère très efficace pour résoudre le problème d'effet domino des outils classiques de la littérature.

L'évaluation d'une configuration s'effectue avec des poids finis ou infinis par l'évaluation des contraintes.

Nous allons illustrer cette contribution avec l'exemple de DIET dans lequel le style architectural de l'application est donc modifié afin d'inclure les contraintes et les mutateurs. L'AC-graphe de DIET doit répondre aux caractéristiques suivantes : chaque composant est déclaré à l'OMNI, chaque LA et chaque SED a exactement un supérieur hiérarchique, son nœud parent dans le sous arbre du graphe, chaque MA et chaque LA gèrent entre $minFilsMA/minFilsLA$ et $maxFilsMA$ / $maxFilsLA$ composants. Dans l'illustration des concepts, ces valeurs sont fixées à 1 et 10, respectivement. Du fait d'hypothétiques restrictions logicielles et d'un nombre limité de machines disponibles, cette architecture ne peut comporter plus de $maxAgents$ agents, dont la valeur est fixée à 100 dans les exemples. Pour exprimer la propriété *d'auto-optimisation*, nous introduisons trois critères :

1. la consommation énergétique. Pour cela, nous utilisons le même modèle que précé-

demment, la consommation est la somme des consommations des machines.

2. la robustesse, c'est à dire la tolérance aux pannes matérielles ou logicielles. La robustesse est un critère non fonctionnel. Elle est ramenée à trois sous-critères fonctionnels se référant, pour chaque service s, à l'ensemble S_s des SEDs le proposant :
   (a) degré de redondance (i.e., cardinalité de S_s). Il s'agit du nombre de SEDs proposant le service s. Plus celui-ci est élevé, moins s est vulnérable aux pannes logicielles affectant un ou plusieurs SED(s).
   (b) dispersion physique (i.e., nombre de machines sur lesquelles un SED $\in S_s$ est déployé). Plus celui-ci est élevé, moins s est vulnérable aux pannes matérielles affectant une machine sur laquelle au moins un SED de S_s est déployé.
   (c) dispersion logique des SEDs au sein de la structure hiérarchique arborescente et étalement physique des LAs sans lien de descendance. Il s'agit ici premièrement de réduire le nombre d'ancêtres communs à chaque couple de SEDs de S_s. Cela réduit la vulnérabilité de s face à une panne logicielle d'un LA. Deuxièmement, il faut que les LAs n'ayant pas de relation de descendance (c'est à dire n'appartenant pas à un même chemin du MA vers un SED) soient déployés sur des machines distinctes. Ainsi, s est plus robuste face à une panne matérielle affectant une machine sur laquelle est déployé un ancêtre d'au moins un SED de S_s.
3. la qualité de service. Celle-ci est évaluée au regard de l'équilibre des charges sur les nœuds de même profondeur. Notons $LA(d)$ *l'ensemble des LAs de profondeur* d, et *M(c) le nombre d'entités gérées par le composant* c. Le critère présent est relatif à la satisfaction de la condition dite d'équilibrage exigeant que, pour toute profondeur d, l'écart type de l'ensemble $\bigcup_{la \in LA(d)} M(la)$ ne dépasse pas un seuil noté maxσ.

Le modèle représentant DIET doit être adapté afin de permettre sa configuration et l'évaluation des trois critères précédemment introduits de manière simple et efficace durant l'exécution. En particulier, les informations relatives à ces critères doivent être facilement accessibles. Il est par conséquent nécessaire de considérer les attributs suivants :

1. Un identifiant unique pour chaque composant.
2. La nature de chaque composant.
3. La profondeur de chaque LA.
4. Le nombre d'entités gérées par chaque composant c de type LA ou MA : $M(c)$,
5. L'ensemble des services fournis par chaque SED et chaque LA. L'ensemble des services fournis par un LA est égal à l'ensemble des services fournis par au moins un SED descendant du dit LA.
6. La machine sur laquelle chaque entité est déployée.

Notation	**Signification**
$LA(d)$	l'ensemble des LAs de profondeur d
$M(c)$	le nombre d'entités gérées par le composant c
$maxSonsMA$ $maxSonsLA$	le nombre maximum d'entités gérées par le MA ou un LA, respectivement
$minSonsMA$ $minSonsLA$	le nombre minimum d'entités gérées par le MA ou un LA, respectivement
maxσ	seuil de la condition d'équilibrage

TABLE 4.3 – Récapitulatif des notations introduites.

Les notations introduites sont résumées dans le Tableau 4.3. La figure 4.15 illustre un AC-graphe représentant une configuration de DIET pour laquelle, l'OMNI n'y est pas représenté.

Les notations présentes dans la figure 4.15 sont récapitulées dans le tableau 4.4.

Notation	**Signification**
$Mach$	l'ensemble des machines disponibles
ID	l'ensemble des identifiants possibles
Nat	l'ensemble des natures possibles d'un composant logiciel
$Lien$	l'ensemble des relations possibles entre composants logiciels ici {"ma2o", "ma2la", "ma2sed", "la2o", "la2la", "la2sed", "sed2o"}, où "o" fait référence à l'OMNI.
S	l'ensemble des services pouvant être assurés
$Serv$	l'ensemble des parties de S
Red	la contrainte de redondance : tout service s est fourni par au moins n SEDs
Loc	la contrainte de dispersion physique : tout service s est fourni par des SEDs déployés sur au moins n machines

TABLE 4.4 – Notations relatives à la caractérisation d'une configuration de DIET

L'application est ici composée de huit composants symbolisés par huit nœuds et leurs relations modélisées par des arcs, tous attribués afin de refléter leurs diverses propriétés. Certains composants, ainsi que le graphe lui-même, sont contraints afin de répondre aux critères introduits. Les contraintes sont représentées à l'intérieur d'un cadre en pointillé, et liées à l'objet qu'elles contraignent par une ligne en pointillé, exceptées celles relatives au graphe lui-même.

Le premier attribut de chaque nœud représente l'identifiant unique, ou nom, du composant

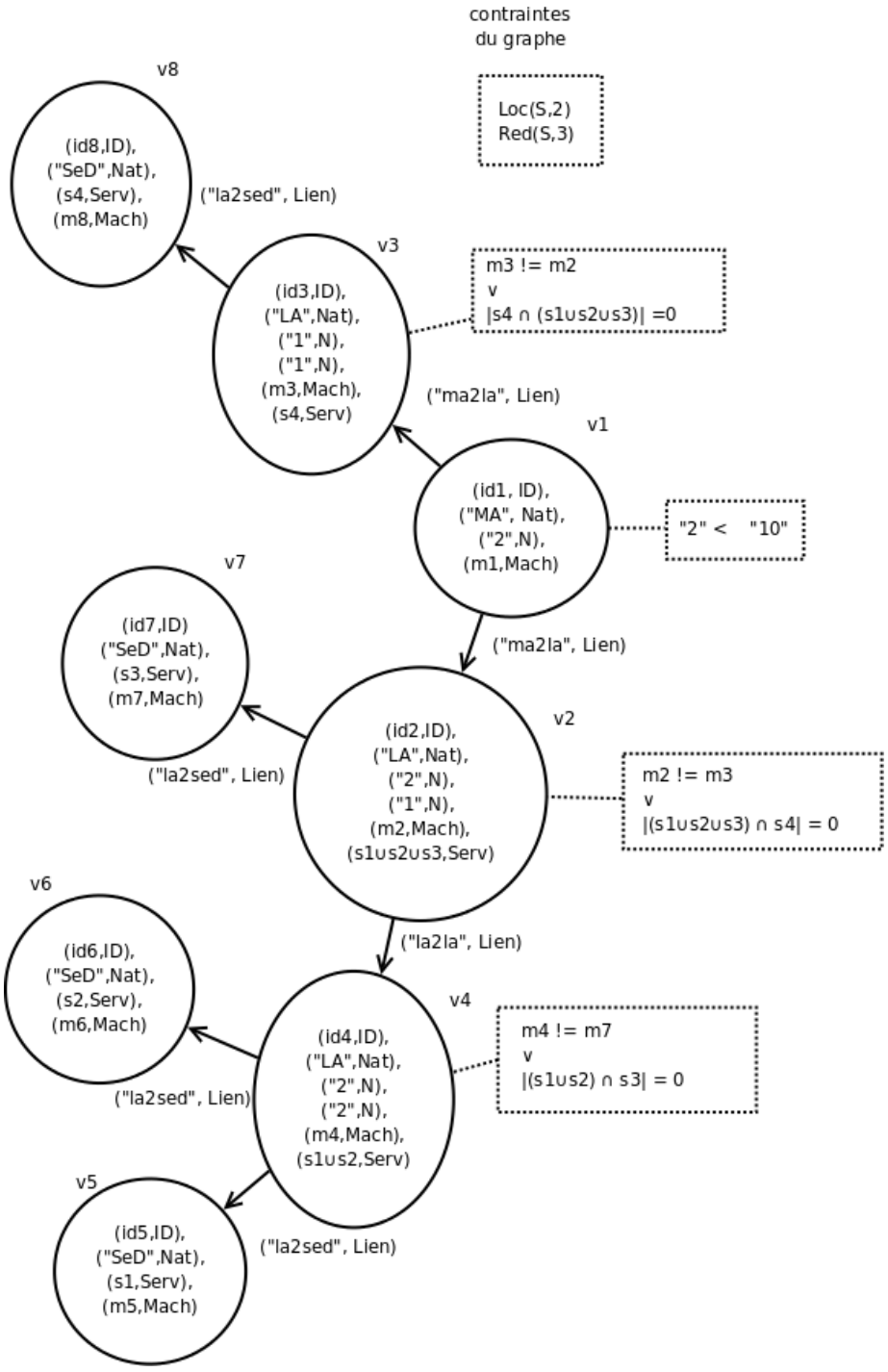


FIGURE 4.15 – Un AC-graphe représentant une configuration de DIET

qu'il représente. Le second, dont le domaine de définition est Nat, reflète sa nature. La configuration comprend un MA nommé id_1. Il gère deux entités et est déployé sur la machine m_1, comme décrit par ses troisième et quatrième attributs, respectivement.

Chaque LA possède deux attributs de plus, liés à sa profondeur et l'ensemble de services qu'il fournit. Dans l'exemple, trois LAs sont déployés. Ils sont respectivement :

- représentés par v_2, v_3 et v_4.
- nommés id_2, id_3 et id_4.
- de profondeur 1, 1 et 2.
- pères de 2, 1 et 2 entités.
- placés sur les machines m_2, m_3 et m_4.
- capables d'accéder aux services $\mathrm{s}_1 \cup \mathrm{s}_2 \cup \mathrm{s}_3$, s_4 et $\mathrm{s}_1 \cup \mathrm{s}_2$.

Finalement, quatre SEDs déployés sur m_5, m_6, m_7 et m_8 fournissent les services s_1, s_2, s_3 et s_4, respectivement.

Le graphe a deux contraintes Loc(S,2) et Red(S,3), reflétant le besoin de redondance et de dispersion physique des services fournis pour améliorer la robustesse du système : cela signifie que tout service de S devrait être fourni par au moins 3 SEDs déployés sur au moins 2 machines différentes.

Le troisième sous-critère de robustesse, la dispersion au sein de l'architecture logique, se traduit par la présence d'une contrainte sur chaque LA. Elle spécifie qu'un LA et ses nœuds frères ne doivent pas être déployés sur la même machine, à moins de ne pas donner accès à des services similaires. Cette contrainte assure que, au sein d'un sous arbre, les SEDs fournissant des services similaires (et déployés sur différentes machines grâce à la contrainte Loc) ne sont pas gérés par des entités elles-mêmes déployées sur la même machine. Ainsi, un plus grand nombre de machines doivent tomber en panne pour qu'un service ne puisse plus être proposé.

Une contrainte sur le MA précise qu'il ne doit pas gérer plus de 10 entités.

Les règles de réécriture sont étendues pour prendre en compte les contraintes et les mutateurs et deviennent des AC-règles. Une AC-règle de réécriture de graphe r_{ac} est un quadruplet (r, ATT, CONS, ACT), où r est une règle de réécriture, ATT et CONS les ensembles d'attributs et contraintes de la règle, respectivement, et ACT un ensemble de mutateurs. L'application de r_{ac} à G selon un morphisme m consiste à appliquer r à G selon m et à appliquer chaque mutateur (algorithme) $\mu \in ACT$.

Les règles de production présentées précédemment pour DIET sont modifiées pour inclure les contraintes et les mutateurs. Ainsi, par exemple, la règle de production consistant à instancier un noeud temporaire géré par un LA en en SED est illustrée figure 4.16.

Cette AC-règle possède aussi le mutateur $\mu_{majServ}(v_a, v_b, \mathrm{i})$: mutateur interne correspon-

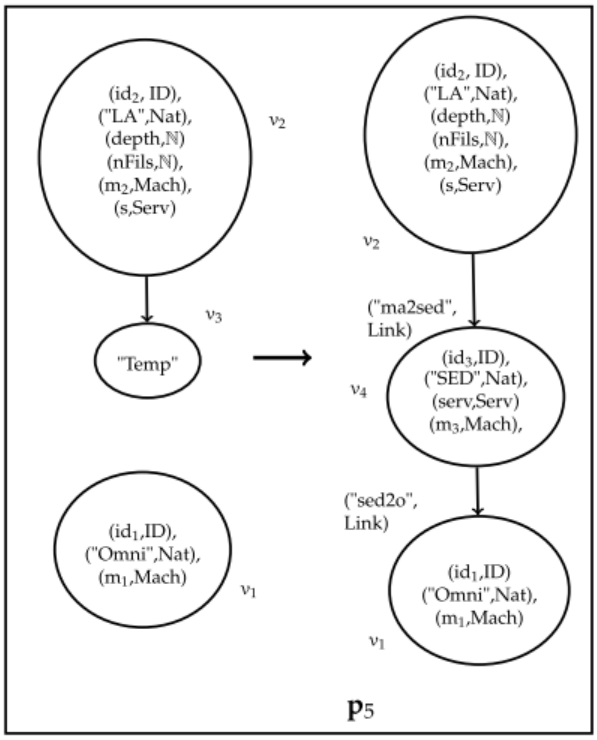


FIGURE 4.16 – Instanciation d'un noeud temporaire géré par un LA en un SED

dant à l'algorithme 6. i est le numéro de l'attribut concerné. Ce processus impacte l'ajout de nouveaux services à $A^i_{v_b}$. L'ensemble des services accessibles depuis son père v_a est alors mis à jour si v_a est un LA (et non un MA). Si une modification est effectivement apportée, elle est récursivement propagée au père de v_a. Dans l'exemple, l'appel du mutateur est : $\mu_{majServ}(v_2,v_4,3)$: l'ajout d'un service à l'attribut de numéro 3 sur le noeud v_4 est impacté dans la liste des services du noeud v_2.

De même un mutateur μ_{inc} permettant d'incrémenter le nombre d'agents dans l'application est utilisé au cours de l'application des règles de production.

Afin de permettre l'évaluation des différentes configurations, nous proposons d'assigner un poids (potentiellement infini) aux contraintes. Ce poids permet de définir une fonction de coût reflétant l'inadéquation d'une configuration. La violation d'une contrainte par une configuration implique l'augmentation de son coût reflétant du non respect ou de l'affaiblissement d'un critère d'évaluation. Plus le poids associé à une contrainte est important, plus elle est critique et plus élevé est le surcoût lié à sa violation. La violation d'une contrainte de poids infini implique une configuration de poids infini assimilable à un état incorrect. Une telle contrainte est alors analytique, une contrainte de poids finie étant qualifiée d'heu-

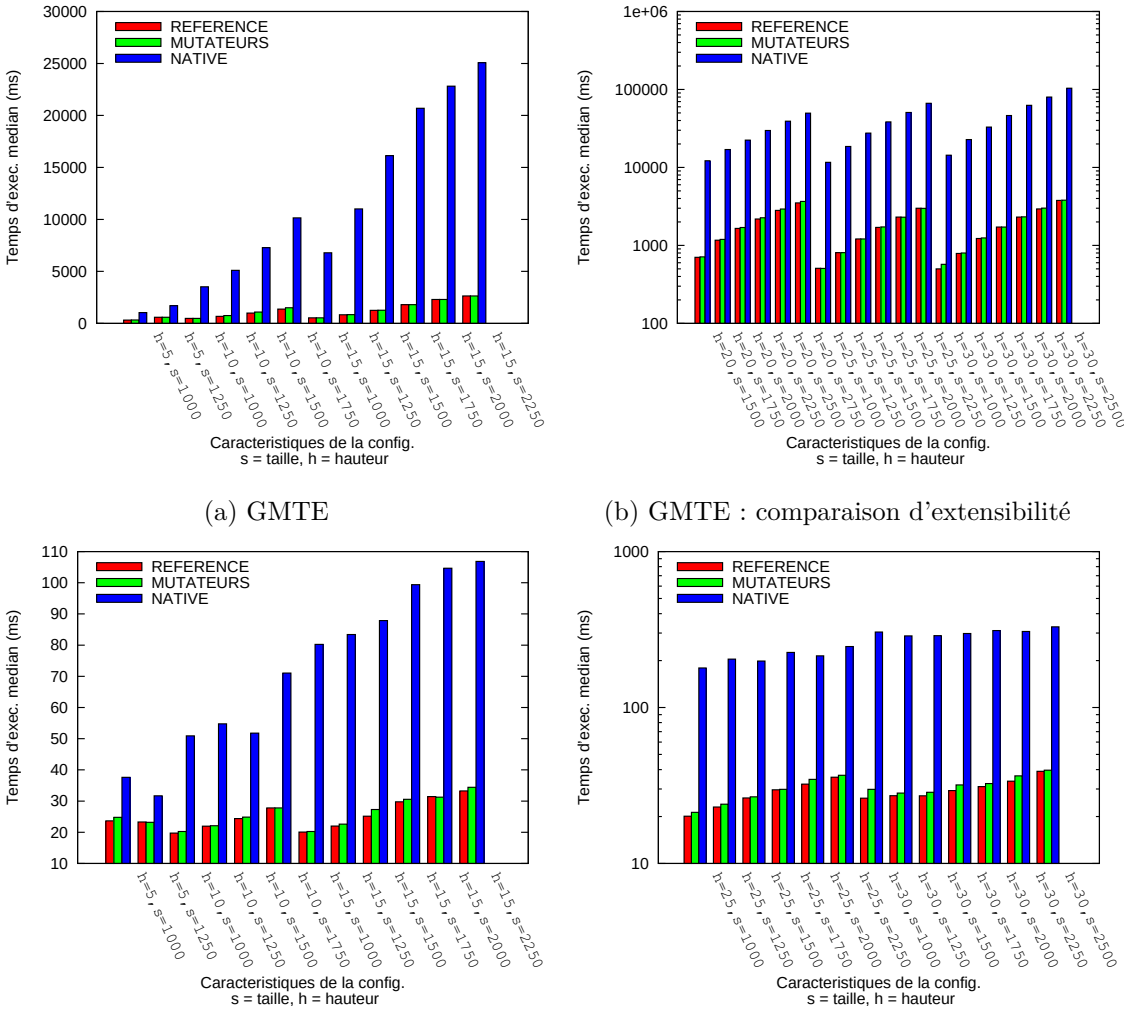
Data: v_a, le nœud attribué à mettre à jour.
v_b, le fils de v_a précédemment modifié.
i, le numéro de l'attribut modifié.
begin
  if $A^2_{v_a}$ = *"LA"* **then**
    oldServ $\leftarrow$ $A^6_{v_a}$
    $A^6_{v_a} \leftarrow$ oldServ $\cup$ $A^i_{v_b}$
    if $A^6_{v_a} \neq$ *oldServ* **then**
      $v_c \leftarrow v \in V$, $(v_c, v_a) \in E$
      $\mu_{majServ}(v_c, v_a, 6)$
    end
  end
end
Algorithm 6: $\mu_{majServ}(v_a, v_b, \text{i})$, Une extension de $A^i_{v_b}$, l'ensemble des services accessibles à partir de v_b, est impacté sur son père v_a.

ristique.

Le coût total d'une configuration est ici calculé comme étant la somme des coûts liés à la consommation énergétique et aux violations de contraintes.

Nous avons expérimenté le surcout des mutateurs par rapport aux approches classiques. Pour cela nous avons considéré l'ajout d'un SED sur un LA de profondeur maximale. Cette reconfiguration implique diverses mises à jour d'attributs :
Premièrement, les attributs représentant le nombre d'agents du système et le nombre d'entités gérées par le LA doivent être incrémentés.
Deuxièmement, l'introduction d'un SED peut étendre l'ensemble des services atteignables depuis un ou plusieurs de ses ancêtres. Dans le scénario exécuté, le SED propose un service inédit ; l'ensemble en question doit donc effectivement être modifié pour chacun de ses ancêtres. Nous considérons ici une architecture DIET complexe comprenant plusieurs MAs, les hauteurs des sous-arbres composés par chacun des MAs, des LAs et SEDs qu'ils gèrent n'ont pas tous la même hauteur et chaque LA non intermédiaire gère au moins 100 SEDs. Nous avons fait varier les configurations étudiées selon deux facteurs : la taille (nombre de composants qui la constituent) et la hauteur (nombre de LAs entre un MA et un SED). Nous avons évalué sur les deux outils de manipulation de graphes AGG [Tae] (très utilisé dans la littérature) et GMTE [Gue07] (outil développé au LAAS) les trois méthodes suivantes :

— "REFERENCE", un référentiel n'incluant pas de modification d'attribut. Il permet d'estimer le surcoût temporel induit par cette modification.
— "NATIVE", la méthode native du moteur, soit une séquence d'application de règles. Premièrement, le SED est ajouté sur un LA dont l'ensemble de services est mis à jour.



(a) GMTE

(b) GMTE : comparaison d'extensibilité

(c) AGG

(d) AGG : comparaison d'extensibilité

FIGURE 4.17 – Temps d'exécution du scénario de reconfiguration

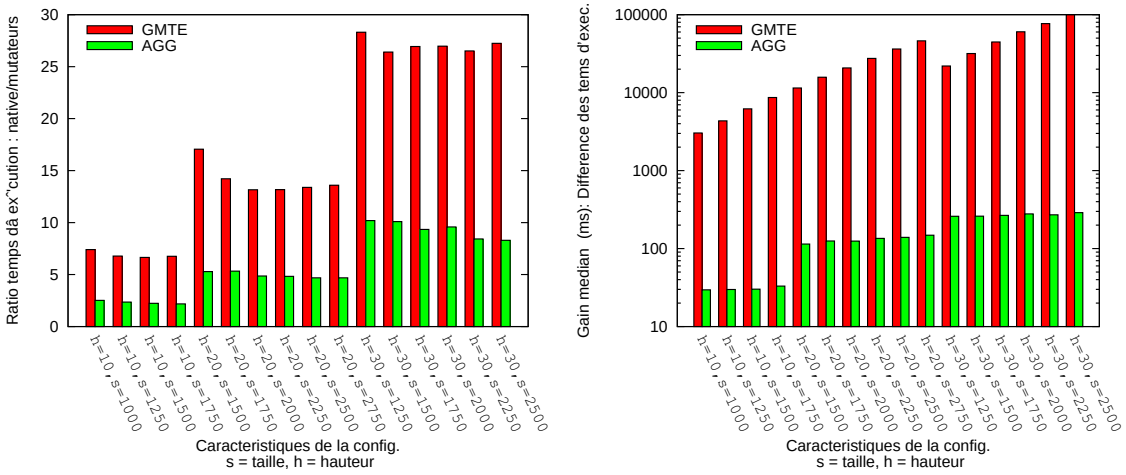
Ensuite, une règle est séquentiellement appliquée afin d'impacter cette modification sur chacun des ancêtres du LA. Seuls les attributs apparaissant dans la règle pouvant être modifiés, chaque application touche un unique LA.

— "MUTATEURS", La méthode proposée ici. Pour ce faire, deux sur-couches ont été implémentées sur AGG et GMTE. Celles-ci mènent à bien l'exécution des règles ajoutant un noeud temporaire sur un LA et l'instanciation du temps temporaire géré par le LA en un SED et leurs mutateurs μ_{inc} et $\mu_{majServ}(v_a, v_b, \text{i})$.

Les figures 4.17a et 4.17c illustrent, pour GMTE et AGG respectivement, les temps d'exé-

cution du scénario de transformation selon les trois méthodes précédemment décrites. Les figures 4.17b et 4.17d montrent l'évolution de ces temps pour de plus grand graphes afin d'apprécier l'extensibilité des méthodes étudiées. Les temps d'exécution rapportés sont les temps médians sur 100 exécutions. Les expériences ont été conduites sur un ordinateur possédant un processeur quadri-cœurs (4M Cache, 2.66 GHz, 1333 MHz FSB) et 8 Go de RAM. Chaque configuration est caractérisée par sa taille et sa hauteur.

Premièrement, les résultats expérimentaux montrent que le surcoût temporel induit par les modifications d'attributs à l'aide de mutateurs (i.e., mutateurs − reference) est petit devant le temps de transformation total si elles sont effectuées à l'aide de mutateurs. Par exemple, pour une configuration de taille 1000 et hauteur 5, ces modifications constituent 7 des 325 ms et 1,2 des 24,8 ms nécessaires à la conduite du scénario à l'aide de GMTE et AGG, respectivement. Ce surcoût augmente linéairement en hauteur et reste globalement invariant vis à vis de la taille du graphe transformé. Pour une configuration de taille 2250 et hauteur 30, il équivaut à 58 ms sur 3010 avec GMTE et 2,7 ms sur 36,6 avec AGG.



(a) Ratio des temps d'exécution

(b) Gain net

FIGURE 4.18 – Méthodes natives vs. mutateurs

Deuxièmement, les mutateurs améliorent significativement l'efficacité et l'extensibilité des modifications du système par rapport aux méthodologies classiques. La figure 4.18a montre l'évolution du ratio des temps d'exécution des méthodes natives sur les mutateurs. Pour une configuration de taille 1750 et hauteur 20, par exemple, ce ratio est d'environ 14 pour GMTE (de 16925 à 1192 ms) et 5,3 pour AGG (de 154 à 28,9 ms). Il augmente avec la hauteur et décroît logarithmiquement en fonction de la taille du graphe considéré. Par exemple, le ratio atteint 27 pour GMTE (de 103796 à 3807 ms) et 8,3 pour AGG (de 330 à 39,6 ms) sur une configuration de hauteur 30 et taille 2500. Contrairement au ratio étudié,

le gain net, i.e. la différence des temps d'éxecution, est strictement croissante à la fois en taille et en hauteur, comme illustré dans la figure 4.18b.

La solution proposée basée sur les mutateurs est meilleure que des méthodes natives d'outils comme GMTE ou AGG. En effet, dans ces outils, mettre à jour les attributs d'un graphe de hauteur H requiert H applications de règles. Une application de règle elle-même ayant des complexités temporelles différentes en fonction des outils. Nous avons pu constaté un effet domino qui était à l'origine de notre réflexion et lié à l'interdépendance entre attributs d'un graphe. En plus des résultats relatifs on peut aussi constater que les résultats absolus sont intéressants (la reconfiguration sur des systèmes à plusieurs milliers d'entités ne dépasse pas les 30 ms sur AGG). Les graphes apportent une modélisation très intéressante pour gérer la cohérence et restent utilisables à grande échelle.

Cette expérimentation illustre à la fois les propriétés *self-optimisation* et *self-configuration* : en effet, l'ajout d'un serveur SED à un LA peut répondre à un évènement de type pic de charge et la contrainte de consommation énergétique relève de l'auto-optimisation du système. D'un autre côté, la mise à jour des services fournis par l'ensemble des SEDs au niveau d'un LA relève de la propriété d'auto-configuration.

Il faut toutefois noter que l'introduction des mutateurs a des conséquences sur la correction par construction. L'inversion d'une règle accompagnée d'une contrainte doit entraîner une post-condition de la règle inverse. Ensuite, cela impose que les mutateurs soient inversibles. La validité de l'inversion était assurée grâce au compteur introduit, or à travers les mutateurs on peut impacter des attributs extérieurs à la règle. Une solution possible serait d'étendre le mécanisme de comptage pour englober et restreindre la modification d'attributs par le biais de mutateurs.

De même la composition de règles est impactée : les mutateurs de la première règle peuvent impacter le graphe et modifier l'applicabilité de la deuxième règle. Ce qui est difficile à prévoir. De plus, si la deuxième règle a des contraintes, il sera nécessaire de trouver un ensemble de contraintes devant être satisfaites avant l'application de la première règle pour que le résultat satisfasse les contraintes d'applicabilité de la deuxième règle.

Les deux approches présentées permettent de mettre en oeuvre les quatre propriétés *self-** des systèmes autonomiques. Nous allons terminer le chapitre par une illustration du *self-optimizing* appliqué à l'infrastructure.

4.5 Reconfigurations dynamiques matérielles : illustration pour du self-optimizing

Nous avons vu au cours des chapitres précédents que souvent les ressources sont surprovisionnées par rapport aux demandes des applications. Cela permet au propriétaire

du datacenter de garantir la performance aux clients mais entraîne des gaspillages de ressources. L'adaptation de ses ressources gérée par des humains n'est pas possible : ce serait trop lent, il y aurait des risques d'erreurs ... L'objectif de notre approche a été d'exploiter les reconfigurations dynamiques autonomes pour optimiser l'utilisation des ressources. Pour cela, nous avons étudié la gestion du réseau d'un datacenter en prenant en compte l'optimisation des coûts énergétiques liés aux équipements réseaux et d'autre part la qualité de service (QoS) fournie aux applications. Nous avons introduit la propriété de "self-optimizing" au niveau matériel en l'appliquant à l'optimisation des coûts énergétiques des datacenters.

Nous avons introduit une approche permettant de décrire un compromis entre, d'une part, la consommation électrique de l'infrastructure réseau et, d'autre part, la dégradation de la qualité de service des applications utilisant ce réseau. Le fait de pouvoir contrôler la reconfiguration dynamique à deux niveaux : au niveau applicatif en reconfigurant dynamiquement des profils de qualité de service et au niveau du matériel réseau avec l'extinction et l'allumage de liens, de modules ou de châssis, permet d'avoir une gestion globale et centralisée du datacenter. En effet, l'utilisation d'un gestionnaire autonomique centralisé permet à l'administrateur de jouer sur le coût énergétique ou sur la performance globale à l'aide d'un seul paramètre contrôlant la politique de haut niveau.

Mesure de consommation réseau Des expériences réelles ont été faites [GCCD+15] sur une plate-forme régionale française appelée GridMip composées de 3 routeurs de base, 3 routeurs de frontière et 150 noeuds.

Le but des expériences était d'évaluer précisément la consommation des équipements réseau pendant le processus de démarrage et le temps nécessaire. Ceci est particulièrement utile si l'on considère des approches qui ont pour but d'éteindre des routeurs pour économiser l'énergie pendant des conditions opérationnelles "légères" tout comme il est nécessaire de prendre en compte la pénalité d'allumage des routeurs.

A l'aide d'un PLOGG wattmeter la consommation d'un routeur de bordure Cisco 7600 composé de 3 modules a été mesuré pendant son démarrage comme indiqué dans la Figure 4.19. Ce routeur consomme 785 watts et 400 secondes sont nécessaire pour l'allumer. La consommation n'est ni constante, ni linéaire ; au lieu de cela nous observons des paliers.

Nous avons également mesuré la consommation d'un module de routeur. Pour cela, nous avons éteint un module 4 x 10Gbps appartenant au routeur de bordure. Nous l'avons ensuite rallumé 20 secondes plus tard. Lorsque le module est éteint, le routeur consomme 195 watts de moins. Quand le module est allumé, il y a deux phases pendant le processus de démarrage : une de 40 secondes à 80 secondes qui consomme 100 watts et une deuxième phase qui consomme 100 watts de plus comme indiqué dans la Figure 4.20.

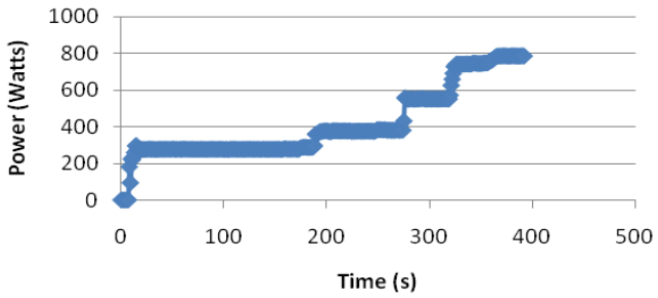


FIGURE 4.19 – Consommation d'un routeur Cisco pendant son démarrage.

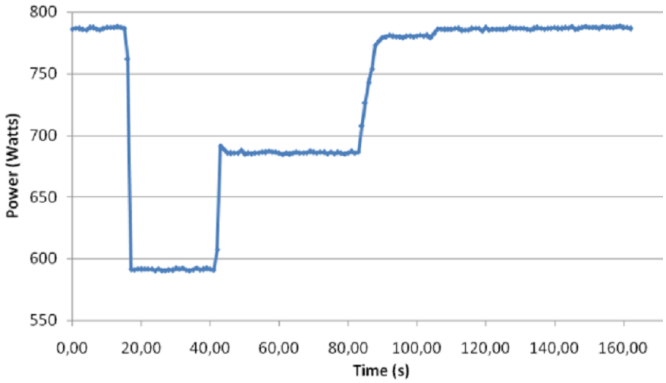


FIGURE 4.20 – Consommation d'un module pendant son démarrage.

Nous remarquons que l'extinction d'un module est réalisée presque immédiatement. On pourrait aussi supposer que démarrage le soit également. Cependant, un module démarre en deux phases, avec le module principal initialisant la mise en marche des ports. L'allumage total dure presque 50 secondes. Finalement la consommation associée aux ports Ethernet a été mesurée comme indiqué dans la Figure 4.21. Pour cette expérience, 18 ordinateurs ont été branchés au module du routeur. L'axe de droite représente le nombre d'interfaces déconnectées (appelé Nb Itf sur le graphique). L'objectif principal de cette expérience était d'observer la consommation avec différents nombres de ports Ethernet connectés.

Nous pouvons constater que la consommation de puissance moyenne d'un port est de 0.94 watts et est linéaire avec le nombre de ports connectés.

Cette étude a été un préambule à la formulation mathématique du problème d'optimisation. Le problème d'optimisation prend en compte la topologie dynamique (extinction des liens, des équipements réseau ou des machines, les capacités des liens et la politique de routage) [SMSB13]. Le problème a été formalisé par un programme linéaire : nous cherchons à

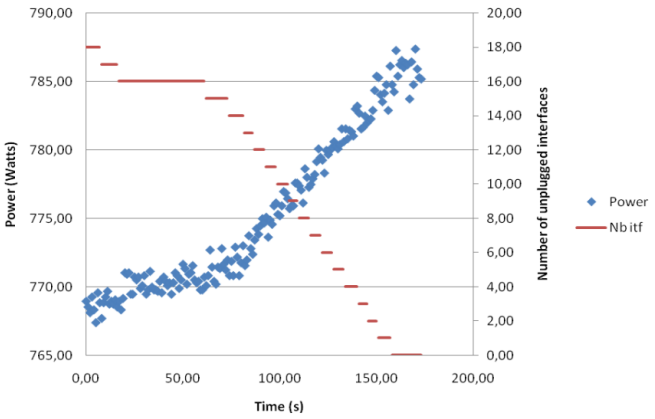


FIGURE 4.21 – Consommation en fonction des ports connectés.

optimiser une fonction objectif linéaire sous un certain nombre de contraintes. Comme nous l'avons dit au chapitre précédent, la programmation linéaire permet de calculer des solutions exactes mais avec des temps de calcul non bornés. Le problème à résoudre est un programme linéaire en nombres entiers plus difficile à résoudre qu'un programme linéaire à variables réelles (car il ajoute des contraintes d'intégrité supplémentaires). Dans cette étude, nous avons résolu le problème à l'aide de la programmation linéaire et à l'aide d'une heuristique basée sur un algorithme glouton. Nous avons comparé les qualités des solutions et les temps de calcul [SMSB13]. Nous détaillons dans la suite la formulation du problème d'optimisation et une partie des résultats.

Formalisation pour la partie réseau. Nous considérons l'ensemble des noeuds réseau appartenant à un même domaine d'administration de la taille d'un réseau de datacenter. La topologie réseau est représentée sous forme de graphe orienté $G = \{\mathbf{N}, \mathbf{E}\}$ de $\mathbf{N}$ noeuds (routeur de bordure, de coeur ou machine) et où les arcs de l'ensemble $\mathbf{E}$ représentent les liens entre les noeuds. Un routeur est composé de plusieurs modules, chaque module a plusieurs ports réseau. Nous considérons que les machines ont un seul module mais potentiellement plusieurs ports. Nous introduisons les notations suivantes :

- $\mid \mathbf{N} \mid = n_N$ et $\mid \mathbf{E} \mid = n_E$
- $c^e_{i,j}$ la capacité du lien (i, j)
- $p^e_{i,j}$ la puissance électrique consommée par le noeud i pour le lien entre i et j
- $z^e_{i,j}$ l'état du port du noeud i pour le lien (i, j), 1 signifiant que le port est allumé, 0 qu'il est éteint
- $p^m_{i,l}$ la puissance électrique consommée par le module l du noeud i
- $z^m_{i,l}$ l'état du module l du noeud i (même convention que pour $z^e_{i,j}$)
- p^c_i la puissance électrique consommée par le noeud i lorsque les modules et les ports

sont éteints (on parle aussi de consommation du chassis)
— z_i^c l'état du noeud i (même convention que pour $z_{i,j}^e$)
— $\mathbf{M_i}$ l'ensemble des modules du noeud i :
$\mathbf{M_i} = \{m_1, \dots m_l \dots, m_{n_{M_i}}\}, \mid \mathbf{M_i} \mid = n_{M_i}$
— $\mathbf{E_{i,l}}$ l'ensemble des ports du module l du noeud i
— $\mathbf{E_i}$ l'ensemble de tous les ports du noeud i, $\forall i \in \mathbf{N}$:
$\mathbf{E_i} = \bigcup_{l \in [1, n_{M_i}]} \mathbf{E_{i,l}}$

La puissance électrique totale du réseau est P_{total} :

$$P_{total} = \sum_{i \in \mathbf{N}} \{p_i^c.z_i^c + \sum_{l \in [1, n_{M_i}]} [p_{i,l}^m.z_{i,l}^m + \sum_{j \in \mathbf{E_{i,l}}} p_{i,j}^e.z_{i,j}^e]\} \tag{4.1}$$

$$= \sum_{i \in \mathbf{N}} p_i^c.z_i^c + \sum_{i \in \mathbf{N}, l \in [1, n_{M_i}]} p_{i,l}^m.z_{i,l}^m + \sum_{i \in \mathbf{N}, j \in \mathbf{E_i}} p_{i,j}^e.z_{i,j}^e \tag{4.2}$$

Lorsque tous les ports d'un module sont éteints, le module peut être éteint :

$$\forall i \in \mathbf{N}, \forall l \in [1, n_{M_i}] : \sum_{(i,j) \in E_{i,l}} z_{i,j}^e = 0 \Rightarrow z_{i,l}^m = 0 \tag{4.3}$$

Ce qui peut aussi s'exprimer avec les contraintes linéaires suivantes :

$$\forall i \in \mathbf{N}, \forall l \in [1, n_{M_i}] : z_{i,l}^m - \sum_{j \in \mathbf{E_{i,l}}} z_{i,j}^e \leq 0 \tag{4.4}$$

$$\forall i \in \mathbf{N}, \forall l \in [1, n_{M_i}], j \in \mathbf{E_{i,l}} : z_{i,j}^e - z_{i,l}^m \leq 0 \tag{4.5}$$

De manière similaire pour les noeuds : si tous les modules du noeud sont éteints, le noeud peut être éteint :

$$\forall i \in \mathbf{N} : z_i^c - \sum_{l \in [1, n_{M_i}]} z_{i,l}^m \leq 0 \tag{4.6}$$

$$\forall i \in \mathbf{N}, l \in [1, n_{M_i}] : z_{i,l}^m - z_i^c \leq 0 \tag{4.7}$$

Lorsqu'un port est éteint, celui auquel il est lié l'est aussi, cette symétrie s'exprime par :

$$\forall i, j \in \mathbf{N} : z_{i,j}^e - z_{j,i}^e = 0 \tag{4.8}$$

Formalisation de la partie application. Nous avons considéré dans notre étude une application comme étant un couple émetteur-récepteur. Une généralisation avec plusieurs récepteurs est possible. De plus nous supposons que la communication n'est que dans un sens, nous négligeons le trafic retour du protocole TCP. Nous avons introduit les notations suivantes :

— A l'ensemble des applications du datacenter :
$\mathbf{A} = \{a_1, \dots a_k \dots, a_{n_A}\}, \mid \mathbf{A} \mid = n_A$
— a_k^s l'émetteur de trafic a_k et a_k^r le récepteur
— N_s^k le noeud émetteur du trafic de a_k^s et N_r^k le noeud récepteur du trafic arrivant à a_k^r
— $\mathbf{NE}$ l'ensemble des besoins en QoS possibles pour toutes les applications. Chaque élément de cet ensemble est lui-même constitué d'un ensemble de valeurs exprimant les caractéristiques de QoS demandées par l'application (débit, gigue, temps de réponse etc) qui sont regroupés dans un tuple. $\mathbf{NE}$ est constitué de valeurs, d'intervalle ou de toutes autres représentations permettant de caractériser un besoin élémentaire en terme de qualité de service. Il est à noter que chaque caractéristique QoS n'est présente qu'une fois dans un tuple. $\mathbf{NE} = \{n_1, \dots n_s \dots, n_{n_{NE}}\}, \mid \mathbf{NE} \mid = n_{NE}$
— $\mathbf{NE^k}$ permet de spécifier pour une application a_k les différents besoins possibles pour cette application.
— $\mathbf{B^k}$ est un ensemble de la même taille que $\mathbf{NE^k}$:
$\mathbf{B^k} = \{b_1^k, \dots b_n^k \dots, b_{n_{B^k}}^k\}, \mid \mathbf{B^k} \mid = n_{B^k}$. Il est composé de variables binaires $b_n^k \in \mathbf{B^k}$:

$$b_n^k = \begin{cases} 1 & \text{si le besoin } n_s^k \ (s = n) \text{ est choisi pour} \\ & \text{l'application } a_k, \\ 0 & \text{sinon} \end{cases}$$

— On suppose qu'il existe une fonction métrique M permettant de mesurer la qualité d'un besoin. Cette dernière permet de définir une relation d'ordre total notée $<$ entre les différents besoins d'une application. Afin de simplifier par la suite la notation, on supposera que les besoins sont rangés dans l'ensemble $\mathbf{NE^k}$ suivant l'ordre $<$ défini avec la métrique M (ceci est toujours réalisable à une permutation prêt sur les indices) :

$$M(n_1^k) < M(n_2^k) < \ldots < M(n_{n_{NE^k}}^k)$$

— $x_{i,j}^k$ on définit un flot $x_{i,j}^k$ comme étant le débit utilisé sur le lien (i, j) par l'application a_k
— AF la fonction AF^k donne pour une application a_k et un besoin choisi n_n^k le débit moyen produit par l'application
— la fonction *RoutingNodes* prend un noeud de départ et un noeud d'arrivée et crée l'ensemble des noeuds de tous les chemins possibles calculés par la fonction de routage du réseau (OSPF, RIP, RIPv2 ...). Cette fonction prend en compte l'extinction et l'allumage des équipements réseau. Pour une application a_k :

$$RN^k = \text{RoutingNodes}(N_s^k, N_r^k)$$

— pour un noeud i, l'ensemble $\mathbf{RN_{input}^{k,i}}$ représente les noeuds connectés à i envoyant du trafic pour a_k et $\mathbf{RN_{output}^{k,i}}$ l'ensemble des noeuds connectés à i et recevant du

trafic de a_k :
$j \in \mathbf{RN^{k,i}_{input}}$ si $j \in RN^k$ et $\exists e_{i,j} \in \mathbf{E}$
$j \in \mathbf{RN^{k,i}_{output}}$ si $j \in RN^k$ et $\exists e_{j,i} \in \mathbf{E}$

Pour une application, un seul besoin peut être choisi à un instant donné, ce qui entraîne la contrainte suivante :

$$\forall k \in [1, n_A] : \sum_{n=1}^{n_{B^k}} b_n^k - 1 = 0 \tag{4.9}$$

Le flot sortant du noeud émetteur est égal au débit moyen produit par l'application et suit la route en fonction du protocole de routage :

$$\begin{aligned} &\forall k \in [1, n_A], j \in \mathbf{E} \text{ so that } \exists e_{N_s^k, j} \in \mathbf{E} : \\ &x^k_{N_s^k, j} - \sum_{n=1}^{n_{B^k}} (AF(n_n^k).b_n^k) = 0 \end{aligned} \tag{4.10}$$

Pour une application a_k le flot qui sort d'un neud émetteur est égal au flot reçu par le récepteur. Nous supposons que les machines sont connectées au routeur de bordure par un seul lien :

$$\begin{aligned} &\forall k \in [1, n_A], \forall i \in \mathbf{E} \text{ so that } \exists e_{N_s^k, i} \in \mathbf{E}, \\ &\forall l \in \mathbf{E} \text{ so that } \exists e_{l, N_r^k} \in \mathbf{E} : \\ &x^k_{N_s^k, i} - x^k_{l, N_r^k} = 0 \end{aligned} \tag{4.11}$$

La propagation des flots s'exprime par les contraintes ci-dessous. Il n'y a pas de perte d'information, pour tous les noeuds qui ne sont pas des machines, la différence entre les débits entrants et sortants est nulle :

$$\forall k \in [1, n_A], \forall i \in \mathbf{E} : \sum_{\substack{j \in \mathbf{E}, \exists e_{i,j} \\ j \neq N_s^k, N_r^k}} x^k_{i,j} - \sum_{\substack{u \in \mathbf{E}, \exists e_{u,i} \\ u \neq N_s^k, N_r^k}} x^k_{u,i} = 0 \tag{4.12}$$

Les flots respectent les capacités des liens :

$$\forall (i,j) \in \mathbf{E} : \sum_{k=1}^{n_A} (x^k_{i,j} - c^e_{i,j}.z^e_{i,j}) \leq 0 \tag{4.13}$$

$$\forall (i,j) \in \mathbf{E}, \forall k \in [1, n_A] : -x^k_{i,j} \leq 0 \tag{4.14}$$

On définit une perte de qualité de service (Quality Loss) QL pour l'application a_k ayant choisi le besoin de qualité de service c (c'est à dire $b_c^k = 1$, tous les autres b_n^k étant égaux à zéro pour $n \neq c$) comme étant :

$$\forall k \in [1, n_A] : QL_{a_k} = M(n^k_{n_{NE_k}}) - \sum_{c=1}^{n_{B^k}} M(n^k_c).b^k_c$$

La perte de qualité de service totale pour le réseau du domaine DS du "datacenter" s'écrit donc :

$$QL_{Total} = \sum_{a_k \in \mathbf{A}} QL_{a_k}$$

Formalisation du problème d'optimisation. Le problème peut s'écrire comme la minimisation d'un critère pondéré par un coefficient $\alpha \in [0; 1]$ qui représente le compromis entre la puissance totale du réseau géré et la perte de qualité de service engendrée. C'est l'administrateur du réseau du "datacenter" qui définit α ; la fonction objectif à minimiser peut s'exprimer ainsi : $\alpha.P_{total} + \beta.(1 - \alpha).QL_{total}$

Ce qui peut s'écrire sous la forme développée et avec les contraintes exprimées précédemment :

$$\underset{x^k_{i,j}; z^c_i, z^m_{i,l}, z^e_{i,j}, b^k}{\text{Min}} \alpha.\{\sum_{i \in \mathbf{N}} p^c_i.z^c_i + \sum_{i \in \mathbf{N}, l \in [1, n_{M_i}]} p^m_{i,l}.z^m_{i,l} + \sum_{i \in \mathbf{N}, j \in \mathbf{E_i}} p^e_{i,j}.z^e_{i,j}\} +$$

$$\beta.(1 - \alpha).\sum_{k=1}^{n_A} (M(n^k_{n_{NE_k}}).b^k_{n_{B^k}} - M(n^k_n).b^k_n)$$

β permet de normaliser les deux critères selon le calcul suivant :

$$\beta = (P_{min} + P_{max})/(QL_{min} + QL_{max})$$

Nous avons résolu ce problème d'optimisation qui est un problème de programmation linéaire avec JOpt[jop05] et le solveur CPLEX[ibm09]. Cette solution optimale représente une maximisation des flots (elle cherche le nombre maximum d'éléments pouvant circuler au sein du réseau tout en respectant les contraintes de circulation), difficilement atteignable dans la réalité, même avec les protocoles de routage dont les coûts des liens sont calculés de manière dynamique [LS06].

En effet, en fonction du routage du réseau, les flots des applications peuvent être divisés par des mécanismes d'équilibrage de charge. Par exemple, dans le cas d'utilisation du protocole OSPF, il faut rajouter des contraintes qui spécifient que les flots sont divisés équitablement sur les chemins de même coût (le coût étant calculé par la métrique OSPF, en général la capacité des liens). Ces contraintes d'équilibrage de charge sont donc spécifiques au

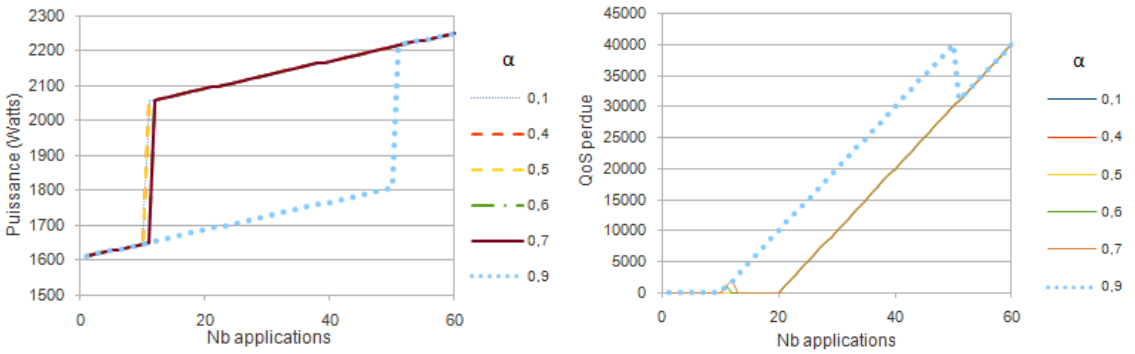
protocole choisi. Nous avons donc proposé une heuristique pour le protocole de routage OSPF largement répandu à l'heure actuelle.

Cela rajoute la contrainte suivante définissant une division des flots équitable entre les chemins de même coût :

$$\forall k \in [1, n_A]; \forall i \in \mathbf{RN^k}, Card(\mathbf{RN^{k,i}_{output}}) > 1, i \neq N^k_s; \forall j \in \mathbf{RN^{k,i}_{output}} :$$

$$x^k_{i,j} = \frac{1}{Card(\mathbf{RN^{k,i}_{output}})} \sum_{l \in \mathbf{RN^{k,i}_{input}}} x^k_{l,i}$$

Cette contrainte est elle-même dépendante d'une variable à minimiser. L'heuristique proposée consiste à trouver une solution d'extinction de liens et à fixer le routage puis à trouver les profils minimum. Dans l'heuristique proposée, basée sur un algorithme glouton, nous trions les châssis (Noeuds), les modules et les ports par nombre inverse d'applications les utilisant, c'est à dire du plus petit nombre de flux au plus grand nombre de flux les utilisant. Nous essayons d'éteindre en premier les ports, puis en second les modules et en dernier les châssis (Noeuds), en respectant pour chacun l'ordre du tri. En effet, cela minimise le nombre d'applications impactées par l'extinction de l'équipement concerné. A chaque extinction nous vérifions s'il existe au moins un chemin OSPF pour toutes les applications et s'il existe une solution. Le même principe est utilisé pour l'extinction des ports ($z^e_{i,j} = 0$), des modules ($z^m_{i,l} = 0$) ou des châssis ($z^c_i = 0$). S'il existe au moins un chemin OSPF pour toutes les applications et que la solution trouvée est meilleure, alors elle est mémorisée, sinon nous continuons d'explorer les solutions pendant max_essais itérations pour essayer d'en trouver une meilleure.



(a) Puissance en fonction du nombre d'applications et d'α dans le cas de routage OSPF avec utilisation de l'heuristique

(b) Qualité de service perdue en fonction du nombre d'applications et d'α dans le cas de routage OSPF avec utilisation de l'heuristique

FIGURE 4.22 – Evolution de la puissance et de la QoS en fonction de α.

Pour la validation, nous avons utilisé la topologie de GridMIP et les valeurs de puissances mesurées dans les expérimentations présentées dans le paragraphe 4.5, nous avons fait varier le nombre d'applications entre 1 et 60 ; chaque application ayant 5 profils (besoins en débit de 400Mbits/s à 1 Gbits/s). Nous avons expérimenté les variations de consommation et de perte de QoS en fonction du nombre d'applications pour différentes valeurs de α (figures 4.22a et 4.22b. Les paliers dans la courbe de puissance correspondent à l'allumage de routeur. Plus α est grand et plus le poids du critère de minimisation de l'énergie est important retardant d'autant l'allumage d'un routeur ainsi pour $\alpha = 0.9$ l'allumage de routeur se fait à partir de 50 applications. La courbe de QoS montre la dégradation de QoS (dans l'expérimentation, cela correspond à une perte de débit) qui augmente avec le nombre d'applications et diminue lorsqu'un routeur est allumé.

La programmation linéaire permet de calculer l'optimal dans un cas où les temps de résolution peuvent être négligeables par rapport au temps de démarrage et d'arrêt des routeurs (qui d'après les tests que nous avons fait sur la plateforme peuvent aller jusqu'à 6 minutes). Une optimisation réseau de ce type doit donc se planifier. Les résultats de l'heuristique suivent ceux de l'optimal avec une route en moins (non utilisable par le protocole OSPF) entraînant une perte de qualité de service plus importante. Des résultats plus complets ont été publiés dans [SMSB13]. Ces travaux ont été réalisés dans le cadre de la thèse de Rémi Sharrock avec l'approche basée sur UML avec une description de la plate-forme et des applications sous forme de diagramme de classes UML. Toutefois, la description pourrait également se faire en utilisant les graphes, ce qui pourrait permettre aussi de garantir que le réseau reste cohérent lors des reconfigurations.

4.6 Projets et contextes applicatifs

Ces deux approches ont été implémentées dans des outils :

— l'approche basée sur UML a été implémenté dans l'outil TUNe (Toulouse University Network), outil développé par Laurent Broto lors de sa thèse encadré par Daniel Hagimont [Bro08]. TUNe [BSB+08] [BHS+08] [Bro08] est un gestionnaire autonomique basé sur un modèle à composants (Fractal). Sa particularité est de permettre d'ajouter des comportements autonomiques à différents types de logiciels patrimoniaux déjà existants. Il offre une vision uniforme par l'encapsulation des logiciels administrés dans des composants. L'administration utilise ensuite l'interface uniformisée offerte par ce modèle à composants et un ensemble de sondes génériques ou de squelettes de sondes réutilisables pour les spécificités du logiciel. Différents modèles étaient de base proposés pour formaliser le style architectural de l'application (diagramme de classes UML), les reconfigurations (diagramme état transition UML) et des wrappers décrivant l'interface d'administration de l'application (méthodes start, stop ....). Les wrappers sont décrits en utilisant un langage dérivé d'XML. Le modèle

résultant est donc un modèle hybride UML-Fractal. Les travaux présentés dans ce chapitre ont permis de remplacer les diagrammes état-transition par les *PDD* afin d'exprimer des reconfigurations plus complexes tout en restant intuitives à décrire.

— l'approche basée sur les graphes a été implémentée dans l'outil GMTE [3] [Gue07] : moteur de recherche d'homomorphisme et de transformation de graphes attribués développé au sein du LAAS-CNRS depuis une dizaine d'années. Implémenté en C++, il ne possède pas d'interface graphique mais est utilisable en tant que librairie C++ ou via une API java. Cette approche a par ailleurs été développée dans le cadre du projet ANR SOP décrit chapitre 3.

4.7 Bilan

4.7.1 Contributions

Dans ce chapitre nous nous sommes intéressés à la formalisation d'architectures logicielles dynamiques : à base de diagramme UML et à base de graphes. L'objectif étant d'exprimer des reconfigurations autonomiques soit matérielles soit logicielles afin de répondre à différents critères d'optimisation. Les deux approches permettent d'exprimer les propriétés *self-** des systèmes autonomiques et ont été illustrées sur différents cas d'application dans [Sha10] et [Eic15].

L'approche basée sur les modèles UML a des limites : dans le style architectural de l'application (diagramme de classes), il est impossible de décrire une récursivité ou plusieurs niveaux variables intermédiaires dans l'application. La gestion de la cohérence peut nécessiter des rollbacks. Néanmoins comme elle a été implémentée dans un outil autonomique, elle a l'avantage de pouvoir gérer un système complet et de reconfigurer de manière globale plusieurs applications en cours d'exécution.

Nous avons introduit la méta-modélisation des Policy Description Diagrams **PDD** qui permettent de décrire un processus de gestion de manière graphique et intuitive pour l'utilisateur final. Nous avons spécifié différents types d'actions et de noeuds de contrôle : les **actions de modification structurelles** ont un impact sur l'architecture du système géré. Elles permettent de rajouter ou de supprimer des instances de certains éléments du système en cours d'exécution, ainsi que de rajouter ou supprimer des liaisons entre ces instances. Les **modifications et sélection de composants** permettent de reconfigurer certains attributs des éléments du système de manière dynamique, de les filtrer suivant certains critères et les **actions d'appel aux méthodes** permettent d'agir sur ces éléments en exécutant des actions sur le système réellement déployé. En ce qui concerne les noeuds de contrôle,

3. *Graph Matching and Transformation Engine*,
homepage : homepages.laas.fr/khalil/GMTE/index.php ?n=GMTE.HomePage

nous avons spécifié des **noeuds de décision** qui créent des chemins d'exécution différents en fonction de conditions exprimées sur des **transitions** sortantes de ces noeuds.

Les mesures présentées dans le paragraphe 4.5 ont montré l'importance de tenir compte de la consommation réseau. Dans l'étude proposée, nous avons formalisé le problème d'optimisation de manière similaire à un problème de placement d'applications sur des machines physiques. La différence étant qu'ici nous choisissons les liens, modules et routeurs pouvant être éteints. Nous avons retrouvé ici le compromis classique entre consommation d'énergie et qualité de service exprimé sous forme de problème multi-critères pondéré. Une des originalités a été de prendre en compte l'impact des protocoles de routage comme OSPF dans l'optimisation. Dans cette étude nous avons supposé que le débit moyen produit par une application était connu. Ce modèle peut s'appliquer pour un centre de données avec des applications relativement persistantes ainsi, les flux de réseau sont prévisibles et ne changent pas radicalement. Ceci est moins applicable aux applications courtes avec un trafic de données éphémère dans le réseau (ce qui peut être le cas dans un cloud).

Le modèle suppose qu'une application a seulement un expéditeur et un récepteur. Cependant, ce modèle nécessiterait d'être étendu pour des applications HPC (par exemple MPI ou MapReduce). Les modèles de communication de type émission/multidiffusion, est plus complexe qu'une application de type client/serveur simple.

Le modèle demanderait donc à être étendu pour une réelle implémentation dans un centre de décision intégrant cette problématique d'optimisation dans ses politiques de reconfigurations autonomiques. Ceci étant, cette étude est un exemple de reconfiguration matérielle tout à fait pertinent mais nécessitant des actionneurs génériques sur les équipements réseaux. Il est cependant nécessaire d'intégrer une modélisation de la plate-forme physique à disposition précisant la topologie réseau [LPP04] par exemple sous forme de *HDD Hardware Description Diagram* (un diagramme de classes représentant l'architecture physique [SSM10]). C'était un travail novateur à l'époque même si cela posait des problèmes au niveau des possibilités fournies par les équipements pour automatiser l'approche et la rendre autonome. Maintenant, les SDN (Software Defined Network) permettrait une implémentation plus adéquate. SDN est un nouveau paradigme réseau permettant la centralisation de gestion de réseau. SDN sépare le plan de contrôle du plan des données des équipements réseau. Ainsi, un routeur est transformé en un dispositif d'expédition simple qui applique des règles envoyées par un contrôleur distant en utilisant un protocole normalisé. Cette approche permet aux administrateurs de réseau d'obtenir un meilleur contrôle sur le trafic dans leur réseau. Ainsi, le contrôleur pourrait résoudre le problème d'optimisation proposé en utilisant des matrices de trafic collectées, calculer la solution de routage amenant le meilleur compromis de QoS/énergie puis mettre en oeuvre la solution trouvée en mettant à jour les règles d'expédition des équipements réseaux et en éteignant les interfaces inutilisées.

L'approche proposée sur les graphes lève des limites des approches UML toutefois, elle a

elle-même des limites. Dans l'approche basée sur UML, la gestion de la cohérence consistait à vérifier que le système satisfaisait un ensemble de propriétés et de contraintes lors de son exécution. Cette vérification pouvant être qualifiée de *model checking on the fly* était réalisée au cours et à la fin de chaque transformation. Cela pouvait entraîner la nécessité de procédures d'annulation.

Dans l'approche basée sur les graphes, la gestion de la cohérence est reportée à la phase de conception : lors de la construction des règles de transformation. Une des contributions majeure de l'approche basée sur les graphes est sa gestion de la cohérence *correcte par construction* : ce qui signifie qu'à partir d'un état correct en suivant des règles de réécriture, nous n'aurons que des états corrects. Toutefois, au cours de son exécution un système peut se trouver dans un état incorrect par rapport à la grammaire. Dans ce cas comment mettre en place la reconfiguration ? Il peut y avoir plusieurs possibilités dont le point commun serait de revenir à un état valide qui devraient être mises en oeuvre au niveau d'un gestionnaire autonomique. La première possibilité consisterait à utiliser les règles réciproques pour supprimer les éléments de l'application rendant l'état invalide. La deuxième possibilité consisterait à trouver le graphe le plus proche possible de l'ancien graphe valide. Cette deuxième solution peut être couteuse en temps de recherche et difficile à calibrer : quel critère utiliser pour le choix du graphe ? La première solution serait la plus simple à mettre en oeuvre ; des reconfigurations visant ensuite à optimiser l'état valide restauré pourraient ensuite être déclenchées.

La deuxième contribution importante de l'approche basée sur les graphes est l'intégration de contraintes et de mutateurs dans la formalisation des systèmes de façon à pouvoir représenter, mettre à jour, évaluer et paramétrer les caractéristiques du système dynamique modélisé. Nous avons par ailleurs montré le gain temporel apporté par cette approche en comparaison aux approches natives [EMS+14].

Les deux approches proposées dans ce chapitre sont génériques : elles permettent de formaliser des reconfigurations indépendamment du type d'application et sans avoir à modifier les exécutables applicatifs. Ceci a été appliqué à différents domaines : les applications parallèles HPC [SKM+09], au Cloud [AHEAMC+14], aux applications M2M (Machine To Machine)[EGM+14]. Il faut noter que l'approche UML sera plus simple à appréhender pour un utilisateur ; toutefois, les outils implémentant les techniques de manipulation de graphes comme AGG sont simples à appréhender car dispose d'interfaces graphiques aidant l'utilisateur. Les règles de réécriture s'expriment visuellement ; le plus compliqué consiste à définir la grammaire. L'outil AGG permet de définir un graphe type pour définir le style architectural et une grammaire, puis de confronter le graphe type à la grammaire. Confronter les deux formalisations permet de vérifier la grammaire.

Les quatre propriétés *self-** n'ont pas le même niveau de difficulté de mise en oeuvre. *self-protecting* est très importante à assurer pendant les reconfigurations ; mais elle est compliquée à mettre en oeuvre. L'approche UML a ses limites pour cette propriété ; par

contre l'approche sur les graphes permet de la vérifier intrinsèquement. La propriété qui est la plus importante pour construire un centre de décision vert autonome est la propriété *self-optimizing* : elle permet d'optimiser le système selon différents critères.

Dans l'idée de la construction d'un centre de décision autonome, un point n'est pas traité dans ce chapitre : c'est le coût d'un reconfiguration. En effet, il serait nécessaire d'ajouter l'évaluation et l'optimisation des reconfigurations elles-mêmes. Lors de processus de décision, il s'agirait alors de comparer le coût de la reconfiguration avec le gain engendré par le passage d'une configuration vers une autre. De même que le chemin pris pour la reconfiguration peut être étudié selon différents critères. Les deux approches se focalisent sur une reconfiguration, il n'y a pas d'optimisation : si plusieurs reconfigurations sont possibles : laquelle serait la meilleure ou laquelle serait la plus pertinente à un instant donné[PRM+12]. De même lors d'une reconfiguration mettant en oeuvre l'ajout d'un composant logiciel, le choix du placement de cette instance n'est pas abordée, elle est déléguée à un système de placement. De plus, nous n'avons pas étudié les compromis entre propriétés *self-**. Tout ceci, nécessiterait la liaison entre l'outil mettant en place les reconfigurations et un module d'optimisation.

Nous essaierons de décrire dans le chapitre suivant comment ceci pourrait se mettre en place.

4.7.2 Encadrement et diffusion scientifique

Ces travaux ont été menés dans le cadre de deux thèses en collaboration avec l'équipe MRS du laboratoire LAAS-CNRS à Toulouse.

- Rémi Sharrock : thèse commencée en 2007 et co-encadrée avec Thierry Monteil du LAAS-CNRS. La thèse a été soutenue en décembre 2010.
- Cédric Eichler : thèse commencée en 2011, co-encadrée avec Thierry Monteil et Khalil Drira du LAAS-CNRS (soutenance en juin 2015).

Plusieurs publications ont été réalisées :

- 2 chapitres de livre : Large-Scale Distributed Systems and Energy Efficiency : A holistic view [13] ; Autonomic Computing and Networking (2009) [14]
- 3 journaux : Software and Systems Modeling (Sosym 2014) [6] ; International Journal of Autonomous and Adaptive Communications Systems (IJAACS 2014) [7] ; International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS 2011) [10]
- 7 conférences internationales : IEEE International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE 2013) [22] ; Energy Efficiency in Large Scale Distributed Systems conference (EE-LSDS 2013) [24] ; IEEE International Symposium on Network Computing and Applications (IEEE NCA 2011) [29] ; International Conference on Autonomic and Autonomous Systems

(ICAS 2010) [32] ; International Conference on Networks (ICN 2010, papier court) [33] ; Challenges of Large Applications in Distributed Environments, Workshop in conjunction with HPDC 2009 (CLADE 2009) [34] ; ACM Symposium on Applied Computing (SAC 2008) [35].

Chapitre 5

Un centre de décision autonome pour la gestion de ressources

5.1 Introduction

Afin d'améliorer l'efficacité énergétique et les ressources en règle général et aussi pour faciliter la gestion des infrastructures, la conception d'un centre de décision autonome semble être nécessaire. Toute la question est de savoir comment faire.

Nous allons tenter dans la suite de ce chapitre de décrire l'approche proposée et de dégager les verrous restant à lever.
Le domaine de recherche de l'efficacité énergétique est une transposition de différentes problématiques recherches comme l'ordonnancement, l'optimisation, les protocoles réseau, l'autonomic computing appliquant des objectifs *green*. Les tentatives de construction de centre de décision autonome dans la littérature se résument à des liaisons entre algorithmes de décision et gestionnaire de cloud. On peut par exemple citer Snooze [Sno],[Fel12].
Snooze est un gestionnaire de cloud, il offre un service de type IAAS (Infrastructure As A Service) (comme OpenNebula ou OpenStack). Il a été conçu et développé par Eugen Feller dans le cadre de sa thèse à l'INRIA Rennes. Snooze gère les VMs en appliquant deux propriétés de systèmes autonomiques : *self-healing* et *self-configuring*. Ainsi, les VMs déployées peuvent profiter des deux propriétés. Néanmoins, les reconfigurations, les propriétés *self-** ne s'appliquent pas aux applications elles-mêmes.
De même nous pouvons citer le *Green Cloud Scheduler* [GCS] proposé dans le cadre du projet européen GAMES [gam]. Un placeur mettant en oeuvre la consolidation interagit avec le gestionnaire de Cloud OpenNebula, les reconfigurations des VMs sont déclenchées grâce à la boucle MAPE-K. Néanmoins comme dans le cas de Snooze, la boucle autonomique ne s'applique pas aux applications.

Le projet ANR Ctrl Green [ctr] s'inscrit dans une démarche d'optimisation énergétique des data centers dans le respect des SLA. Ce projet développe une approche basée sur la programmation synchrone et cherche à coordonner plusieurs boucles de contrôle autonomiques et multiples. Dans [ERDPB+13], les auteurs proposent l'intégration d'une boucle de contrôle dans la boucle d'Entropy [HLM+09], un gestionnaire open-source de consolidation. Ceci leur permet de fournir un service de consolidation élastique qui provisionne dynamiquement les ressources nécessaires.

Ce que nous proposons est une intégration des décisions *energy aware* dans les outils autonomiques qui reconfigurent les applications afin de les rendre auto-gérées. Ainsi, les applications intègreraient des préoccupations *green* notamment à travers la propriété *self-optimizing*. En fonction par exemple d'un événement détectant le dépassement d'un seuil de température, une reconfiguration adéquate pourrait être mise en oeuvre.

Nous proposons de construire un centre de décision autonome incluant des politiques pour le passage à l'échelle ou la disponibilité des applications tout en gérant les compromis entre performance, disponibilité et énergie. Le lecteur a bien compris que notre contribution ne se situe pas dans le développement d'outils de gestionnaire autonomique mais plutôt dans leur utilisation et dans la proposition de formalisme de description de style architectural et de politiques de reconfiguration de haut niveau. Nous avons vu au chapitre 4 que les reconfigurations peuvent aussi s'appliquer au niveau matériel. En reliant ces politiques à un système de décision (c'est à dire un ensemble d'algorithmes d'optimisation multi-objectifs comme ceux proposés au chapitre 3), nous obtenons des centres de calcul verts et autonomes.

5.2 L'approche proposée

5.2.1 Les infrastructures ciblées

Dans le cadre des travaux de recherche exposés dans ce mémoire différentes infrastructures ont été étudiées.

Dans le cadre du projet SOP, nous avons appliqué les algorithmes proposés au chapitre 3 au multi-cloud en les complétant par un méta-placeur dont le rôle était de sélectionner le cloud le plus adapté pour répondre à la requête utilisateur en se basant sur des métriques agrégées.
Au niveau d'un Cloud toujours dans le projet SOP nous avons proposé un algorithme de réallocation de machines virtuelles basé sur le principe de la consolidation en intégrant les surcoûts des leviers verts : migration de VM, et *on/off* des machines physiques (algorithme SOPVP). Nos partenaires [ABP11] ont montré qu'une approche décentralisée

non-coopérative donnait de bons résultats comparé à une approche centralisée. Nous nous focaliserons donc sur une instance de décision sachant que l'on pourrait les dupliquer pour des plates-formes large échelle. De même que, le framework MREEF présenté dans le chapitre 2 peut s'utiliser de manière centralisée où une instance est en charge de l'ensemble de l'infrastructure ou de manière décentralisée où chaque noeud est en charge de sa propre supervision.

Dans le cadre du projet CoolEmAll, c'est au niveau d'un centre de calcul que nous avons proposé des algorithmes de décision tenant compte de considérations énergétiques des serveurs de calcul et du système de refroidissement. Nous avons vu l'importance de la gestion de l'hétérogénéité des machines notamment pour la gestion de la chaleur. Une étude statique du placement des machines dans les racks sera importante. Cela nous a conduit à proposer des heuristiques de placement en utilisant des algorithmes gloutons comme pour les infrastructures de type cloud.

Notre approche combinant des architectures hétérogènes pour constituer un datacenter à consommation énergétique proportionnelle est innovante. La transposition du concept Big.LITTLE a montré son efficacité pour utiliser de manière adéquate les ressources hétérogènes afin d'obtenir une consommation d'énergie proportionnelle à la charge. Cette approche peut s'appliquer à de petits datacenters. La tendance a été à la mutualisation des coûts d'infrastructures et à la concentration des ressources dans de grands centres de calcul. On a pu constaté des conséquences en terme d'impact écologique/énergétique (concentration des datacenters en Caroline du Nord par exemple et ses conséquences sur le réchauffement climatique du à l'utilisation du charbon pour la production d'électricité). Des travaux de recherche tendent à proposer plutôt une distribution des ressources sous forme de fédération de clouds [BDF+14], [BRC10]. Nous sommes donc persuadés que notre approche proportionnelle sera utile dans cette perspective de distribution des ressources.

Des caractéristiques spécifiques des applications distinguent les infrastructures *HPC*, des infrastructures *cloud*. Dans le contexte HPC, il y a une tâche par machine physique, les objectifs des algorithmes de placement concernant la qualité de service sont principalement le *makespan* des tâches ou éventuellement des contraintes de date de fin à respecter. En revanche, dans le contexte cloud, il y a plusieurs machines virtuelles par noeud physique. Les critères de qualité de service sont souvent plus complexes : temps de réponse (interactivité avec l'utilisateur), budget (de l'utilisateur, de l'opérateur) ... Potentiellement, des leviers peuvent avoir des actions contradictoires sur les applications. Il est alors nécessaire de considérer le système complet sans connaissance des applications comme il a été proposé dans le chapitre 2.

Ainsi, finalement le type d'infrastructure n'aura pas de conséquence sur l'utilisation du centre de décision tel qu'il est proposé ici si ce n'est sur les objectifs des algorithmes de décisions. Tout l'intérêt d'un centre de décision autonome sera dans sa capacité à gérer de manière optimale et dynamique (c'est à dire à la volée) l'infrastructure, les ressources et

les applications.

5.2.2 Les modèles/concepts à appliquer

Une gestion dynamique de l'infrastructure est nécessaire en utilisant tous les leviers verts à disposition :

- exploiter le DVFS en fonction du contexte d'exécution et les différents leviers verts matériels disponibles pour tous les composants matériels d'une infrastructure : processeur (DVFS), carte réseau (LPI), disques ...
- on/off de machines et de liens réseaux
- live-migration de machines virtuelles.
- utiliser au mieux l'hétérogénéité des machines pour avoir une consommation d'énergie proportionnelle

Nous proposons pour cela de nous appuyer sur les concepts étudiés dans les chapitres précédents en :

- optimisant le placement des applications sur les ressources disponibles : les critères de placement doivent être multi-objectifs : énergie, chaleur, QoS
- exploitant la ré-allocation : réévaluation régulière du placement pour tenir compte des arrivées/départs de tâches, et de manière général du dynamisme de l'exécution. Ceci ne peut se faire qu'en utilisant le levier migration des machines virtuelles et cela peut permettre de consolider la charge sur un sous-ensemble de la plate-forme. Suite à cela, il sera possible d'exploiter le levier extinction de machines (ou suspend to ram).
- en mettant en oeuvre une reconfiguration et adaptation du système. Ceci peut se faire à différents niveaux : au niveau du système complet en actionnant les leviers verts adaptés comme le fait MREEF, au niveau matériel et au niveau des applications en appliquant les propriétés *self-** des systèmes autonomiques.
- dans tous les cas, cette gestion dynamique des ressources et activation des leviers ne peut se faire qu'en prenant en compte les surcouts engendrés. Sachant qu'un datacenter en 2014 avait en moyenne un PUE de 1,7 ; il est important que le système de décision prenne en compte toutes les origines de consommation d'énergie : calcul et refroidissement.

Nous sommes convaincus qu'en considérant des reconfigurations des applications nous gagnerons en efficacité. Nous avons présenté au chapitre 4 deux approches de formalisation de politiques de reconfigurations de haut niveau : une approche basée sur UML et une approche basée sur les graphes. Nous avons discuté leurs avantages et limites. Nous pensons que pour une utilisation dans un centre de décision, une approche hybride théorie des graphes/ UML pourrait permettre une gestion dynamique des applications. En effet, la théorie des graphes permet l'expression des actions de reconfiguration sous forme de règle de réécriture, cela nous apportera la garantie d'avoir des reconfigurations préservant la

cohérence du système. Des politiques UML permettront d'exprimer les enchaînements de règles de réécriture afin de formaliser de façon intuitive les reconfigurations des différentes applications s'exécutant sur l'infrastructure en utilisant les règles de réécriture de graphe des applications déployées. UML nous apportera une souplesse de description permettant d'optimiser des reconfigurations de plusieurs applications se partageant une même infrastructure. UML sera utilisé au niveau des reconfigurations multi-applications et la théorie des graphes au niveau mono-application.
Les reconfigurations ne peuvent se faire indépendamment des décisions de placement, un dialogue et une interaction sont donc nécessaires entre le gestionnaire autonomique en charge des reconfigurations et le placeur.

5.2.3 Les briques nécessaires

Une ébauche d'autonomic manager intégrant des problématiques *energy aware* a été décrite dans [BSDCP10] et une première preuve de concept a été mise en oeuvre dans le projet SOP [BCD+15]. Nous avons en effet mis en place une liaison entre le placeur SOPVP et l'autonomic manager (outil Frameself [BAM13]) pour gérer le cas des migrations inter-cloud.

L'architecture proposée dans le projet SOP a été la suivante (elle est décrite dans la figure 5.1) :

- un gestionnaire de cloud (OpenNebula),
- une extension des politiques de placement du gestionnaire de cloud OpenNebula (SOPVP) pour intégrer les contraintes d'énergie et de qualité de service,
- un logiciel pour fédérer les différents clouds : Vishnu et un algorithme de meta-placeur,
- un *market place* pour gérer l'utilisation des ressouces des utilisateurs de manière *green* : des eco-points sont attribués aux utilisateurs ayant favorisé l'utilisation communautaire des ressources permettant ainsi des économies d'énergie.
- un autonomic manager pour permettre une gestion autonomique de l'architecture distribuée.

Nous proposons ici d'étendre cette architecture en détaillant ce que devraient être les interactions entre le placeur et l'autonomic manager, en intégrant une gestion du système complet avec un brique logicielle qui correspondrait à MREEF.

Les différentes briques nécessaires présentées figure 5.2 sont les suivantes :

- *un gestionnaire de cloud.* Il s'agit d'un ensemble de logiciels permettant de déployer des infrastructures de cloud computing. On peut citer OpenNebula ou OpenStack. Les deux offrant une architecture modulaire permettant de contrôler les différentes

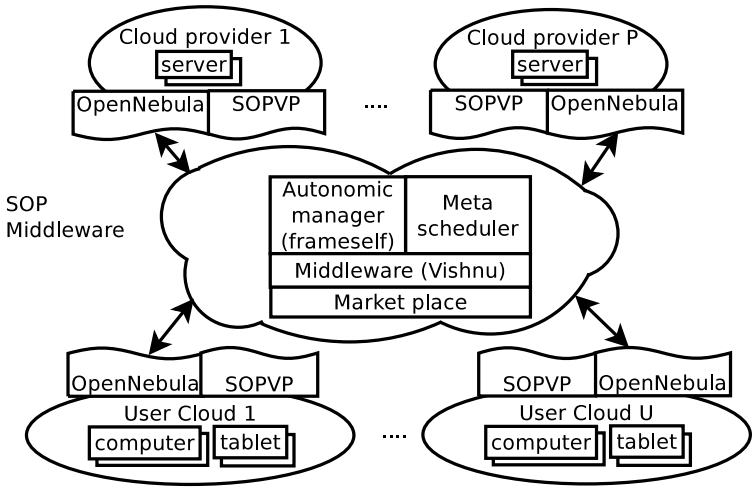


FIGURE 5.1 – Architecture SOP

ressources des machines virtuelles et de mettre en oeuvre des *live-migration* ou différentes actions sur les VMs. Dans la figure 5.2 seules les parties responsables du placement et des *live-migration* sont présentées. En effet, pour la mise en place d'un centre de décision autonome, il sera nécessaire de modifier le placeur par défaut du gestionnaire de cloud. Nous le considérons comme une brique en tant que telle et la décrivons ci-après.

— *un placeur.* Il s'agit de proposer deux types d'algorithmes de placement : un placement initial des VMs et une réallocation périodique afin de régulièrement optimiser l'affectation des ressources. Les algorithmes de placement devront prendre en compte différents objectifs : consommation d'énergie, qualité de service (le perpétuel compromis entre énergie et dégradation de performance acceptable) et la température. Les différents algorithmes proposés au chapitre 3 pourraient être implémentés (nous rappelons que SOPVP a été implémenté dans OpenNebula). Les algorithmes de placement seront donc multi-critères et l'on pourrait les qualifier d'*energy et thermal aware*. Dans le cas d'un centre d'une infrastructure de petite taille, l'approche proportionnelle du chapitre 2 pourra être utilisée. Le placeur sera dans ce cas en charge du choix de l'architecture (Little, Big, Medium) adéquate pour répondre aux besoins des VMs. La mise en place de l'élasticité adéquate de l'application sera mise en oeuvre par le gestionnaire autonomique.

— *un gestionnaire autonomique.* C'est la brique logicielle qui sera en charge de la mise en oeuvre de la boucle MAPE-K. Nous pourrons ainsi décrire le style architectural des applications déployées. Le gestionnaire autonomique sera en charge de détec-

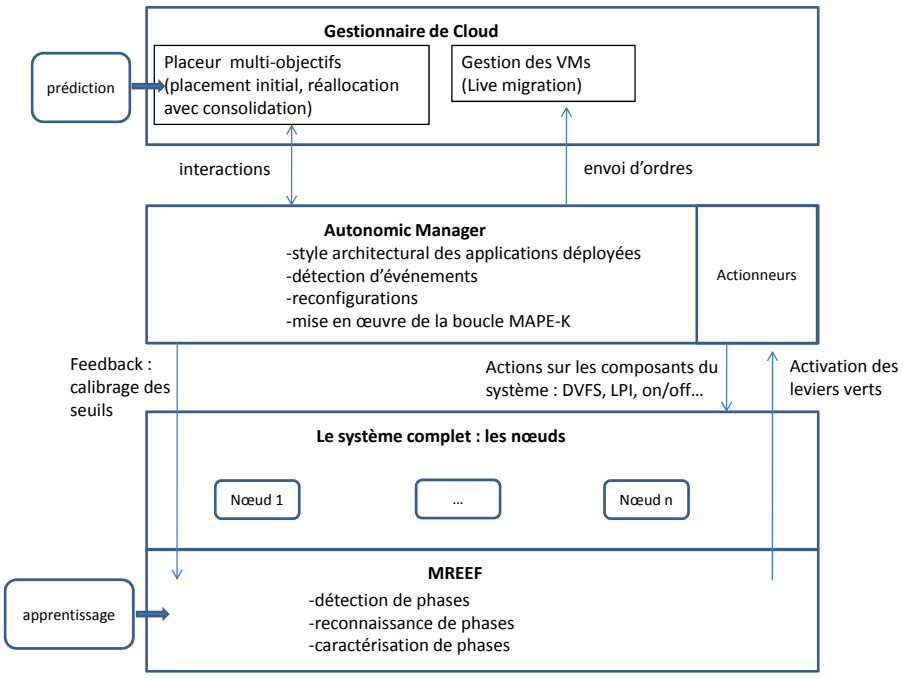


FIGURE 5.2 – Architecture et interactions pour un centre de décision autonome

ter des événements et déclencher les reconfigurations dynamiques autonomes. Le système d'administration autonomique garde une trace de l'architecture physique et logicielle de l'application. A la réception d'une notification suite à une action d'un utilisateur ou provenant d'une sonde, il est chargé de prendre une décision s'il y a lieu et d'agir en conséquence c'est à dire de mettre en oeuvre les actions sur les leviers. Dans le chapitre 4 nous avons illustré la propriété *self-optimizing* avec l'application DIET néanmoins, les politiques de reconfigurations ne peuvent devenir très puissantes et optimales sans faire appel à un placeur. Nous proposons donc ici de lever ces limites en permettant au gestionnaire autonomique un dialogue avec le placeur permettant de prendre des décisions dynamiques. Le gestionnaire autonomique sera en charge de la mise en oeuvre des reconfigurations en utilisant les actionneurs. Cela entraînera une interaction avec le gestionnaire de cloud.

— *un gestionnaire du système complet.* Il s'agit ici d'intégrer dans une brique logicielle une gestion du système complet comme il a été présenté dans le chapitre 2 en uti-

lisant par exemple MREEF. En effet, cette approche apporte une vue globale et bas niveau du système permettant de s'abstraire des applications et surtout sans connaissances sur elles. Il serait en effet illusoire de supposer que nous possédons pour toutes les applications des informations nécessaires pour mettre en oeuvre toutes les fonctionnalités des briques précédentes. Dans le pire des cas, MREEF sera donc la brique logicielle permettant de trouver quelles actions sont utilisables pour reconfigurer l'infrastructure dans son ensemble sinon, cette brique permettra de compléter les actions logicielles mise en oeuvre par les briques précédentes en utilisant les leviers bas niveaux des différents composants et en intégrant de manière globale les interactions entre les différents workloads en exécution.

Le centre de décision proposé repose sur différentes interactions entre les briques logicielles. Nous ne décrirons pas les interactions entre le placeur et le gestionnaire de cloud car le placeur doit être intégré au gestionnaire de cloud en remplaçant le module par défaut.

Interactions entre le gestionnaire autonomique et le gestionnaire du système complet. Dans sa thèse, Landry Tsafack [Che13] a proposé une architecture pour MREEF reposant sur deux composants : le coordinateur et le module de mise en oeuvre. Le coordinateur exécute des tâches de profilage de système incluant la détection de phase, la caractérisation de phase et l'identification de phase. Il notifie le module de mise en oeuvre des décisions de reconfiguration pour adapter le système quand une phase a été identifiée. Ce module capture les métriques d'utilisation de ressources en consultant les compteurs et prend des décisions de reconfiguration de système.

Dans le cadre de la mise en place d'une infrastructure comme proposée figure 5.2, la mise en oeuvre des reconfigurations relèverait du gestionnaire autonomique. Ce serait l'outil autonomique qui serait en charge des actions sur les effecteurs (mise en oeuvre réelle des leviers verts). En effet, le gestionnaire doit avoir une vision globale de l'état des applications et des infrastructures notamment pour informer le placeur des modifications de ressources. On pourrait compléter l'interaction entre MREEF et l'outil autonomique en mettant en place un mécanisme de feedback vers MREEF pour éviter des oscillations (activations des leviers trop fréquentes et inversées) et utiliser la boucle autonomique pour calibrer le seuil de détection de phase.

Interactions entre le gestionnaire autonomique, le placeur et le gestionnaire de Cloud. On parlera ici d'application pour désigner l'exécutable que l'utilisateur souhaite lancer qui peut être composée de différents éléments (composants de l'application, par exemple, les MA, LA, SED dans le cas de l'application DIET) qui seront encapsulés dans des machines virtuelles.

Le placeur est en charge de trouver le placement des VMs et le gestionnaire autonomique grâce à la propriété *self-configuring* prépare tous les fichiers de configuration nécessaires au déploiement et appelle le gestionnaire de cloud qui lance les VMs.
Lorsque le placeur consolide les VMs et génère des migrations, le gestionnaire autonomique est averti et c'est lui qui orchestre la mise en oeuvre en faisant des appels au gestionnaire de Cloud.
Lorsque le gestionnaire autonomique détecte des événements nécessitant des décisions, le placeur est appelé. Cela peut se produire dans des cas de *self-healing* ou *self-optimizing* par exemple lorsqu'un seuil de QoS est dépassé, lorsqu'une sous-charge, une sur-charge ou un pic de température, un pic de consommation d'énergie sont détectés. Le placeur résout alors les optimisations nécessaires et délègue au gestionnaire autonomique la réalisation du plan d'actions.

5.3 Les verrous à lever

5.3.1 Leviers verts

Certains leviers verts sont dans certains contextes difficiles à mettre en oeuvre. Nous détaillons ici quelques exemples.

Live-migration Dans le cadre du projet SOP, dans un contexte multi-cloud nous nous sommes heurtés au problème de la migration inter-cloud. En effet, les différentes clouds gérés par différents *gestionnaire de cloud : OpenNebula* ne pouvaient gérer cette migration. Des technologies comme le SDN sont une solution utilisable pour lever ce problème technique.
La migration entre architectures hétérogènes n'est pas simple non plus. Des évolutions technologiques sont en cours pour gérer la migration inter-architectures.
Enfin, nous avons pu voir que les coûts de migration sont nécessaires à prendre en compte dans nos approches. Nous n'avons pas cherché dans nos travaux à améliorer la performance de la *live-migration* mais des travaux existent. On trouve dans la littérature différentes façons de réduire le temps de migration. La gestion de la mémoire est un aspect important qui peut être amélioré, par exemple en utilisant des noeuds de stockage (Storage Area Network) [AHTH14], ou en améliorant les pré-copies [HLS+14].

Mémoire, disque et réseau Des améliorations au niveau de certains leviers verts système sont souhaitables. Pour être opérationnel, le système d'exploitation doit être capable d'éteindre des barrettes mémoire. Au niveau du disque, les évolutions matérielles permet-

tront aussi de gagner en efficacité. La modification de bande passante des cartes réseau n'est toujours bien supportée par le matériel actuel, l'activation du levier peut être longue.

On/off Le levier allumage/extinction de machines physiques est un levier important. En effet, il permet d'économiser l'énergie consommée lorsque les machines ne sont pas utilisées. Nous avons vu que lorsqu'il est couplé à un algorithme de consolidation et une politique de gestion des machines minimisant les surcoûts d'allumage et d'extinction engendrés on pouvait obtenir des gains d'énergie. Néanmoins, les industriels accepteraient ils des reconfigurations autonomiques de leurs datacenters déclenchant des allumages/extinction des machines autonomes et dynamiques ? Confrontés au problème de rendement de leur infrastructure et de rentabilité économique, ne seraient ils craintifs ? En effet, ils sont liés à des contrats avec les clients les engageant à respecter des critères de disponibilité. Afin de favoriser une meilleure acceptation des propriétaires des centres de calcul, il serait envisageable de proposer les actions à mettre en oeuvre. Pour optimiser les gains d'énergie, il parait important d'inciter les utilisateurs à utiliser les ressources quand elles sont libres. Il faudrait donc accompagner la mise en place d'un tel centre de décision autonome par un travail auprès des utilisateurs d'une part auprès des administrateurs systèmes des centres de calcul pour qu'ils acceptent une telle solution d'autre part auprès des clients pour envisager de transformer/adapter leurs usages.

5.3.2 Monitoring et profilage

Le gestionnaire autonomique repose sur la détection d'événements donc sur un système de monitoring qui consommera lui même des ressources et qui pour être pertinent devra être suffisamment rapide pour que les reconfigurations soient toujours efficaces lorsqu'elles seront enclenchées. Dans sa thèse Rémi Sharrock [Sha10] a montré la possibilité de mettre en place des sondes agrégées chargées de surveiller plusieurs composants de l'application logicielle. Dans le cas d'une application hiérarchique comme DIET, dans le cadre d'une expérimentation sur Grid'5000 de *self-healing* où nous simulions une panne de cluster contenant les SEDs d'un LA, l'agrégation d'événements (détection de la panne de tous les SEDs permet de les réparer en parallèle) et d'obtenir un temps moyen de réparation du cluster de 15012 ms contre 53940 ms dans le cas non agrégé. En contre-partie, il y a une très légère perte de réactivité car la sonde doit attendre d'agréger suffisamment d'événements avant d'envoyer une notification de réparation à l'autonomic manager. Les problématiques de pertinence et d'interprétation des événements peuvent aussi être résolues en ajoutant du sens aux événements. Nous y reviendrons dans le paragraphe suivant.

Certaines approches nécessitent un bon profilage de consommation de ressources des applications : consommation de puissance, de calcul ... De même les capacités de calcul et

la consommation de puissance des machines de l'infrastructure doivent être connues. Pour cela, les techniques de prédiction et d'apprentissage peuvent être utiles pour compléter les approches.

5.3.3 Apport de la sémantique sur les décisions

Il s'agirait d'associer du sens aux événements reçus en utilisant par exemple des ontologies et des descriptions des événements, exprimées respectivement en OWL et RDF, qui sont les formalismes promus par le W3C dans le cadre du développement du Web Sémantique. Une ontologie est une description formelle explicite des concepts, c'est une façon de représenter la connaissance indépendamment du système qui les utilise. Les langages de représentation des ontologies sont issus entre autres des formalismes graphes.

Cette sémantisation de données permettrait ensuite de prendre des décisions adéquates. Le gestionnaire autonomique serait ainsi capable de :

- prioriser les événements,
- de détecter des incohérences entre les événements reçus (par exemple parce qu'un événement est devenu périmé),
- de détecter des incohérences dans les événements reçus par rapport à ce que l'on pouvait attendre d'après la description des capteurs,
- de mettre en relation plusieurs événements en fonction de la sémantique associée.

Les décisions de reconfiguration seraient ensuite plus précises et bien optimisées : elles seraient notamment plus rapides. Sur ce point, nous avons participé au dépôt d'un projet de recherche avec notamment l'équipe MELODI de l'IRIT avec laquelle nous aimerions investiguer cette étude.

5.3.4 Apprentissage et prédiction

Lorsqu'un système doit réagir dynamiquement et de manière autonomique, la prédiction de ce qui devrait arriver dans un futur proche et un apprentissage des bonnes réactions aux différents événements pouvant se produire sont des pistes qui peuvent rendre le système encore plus efficace. Dans la brique du gestionnaire du système complet, l'approche MREEF intègre dans la phase *Classification de phase et adaptation système*, une prédiction simple consistant à prédire le comportement du système au prochain pas de temps. Pour ceci, nous avons utilisé un mécanisme très similaire aux algorithmes de cache qui consiste à considérer que le comportement du système à $t + 1$ sera le même qu'à l'instant t. Ce mécanisme très simple gagnerait à être amélioré.
La charge du système peut elle aussi être prédite. C'est d'ailleurs une des hypothèses de notre approche d'infrastructure hétérogène. La prédiction de charge est un domaine de recherche en soi et nous n'avons pas la prétention de développer notre propre module de

prédiction. Plusieurs techniques peuvent être utilisées pour faire des prédictions basées sur des modèles différents comme le *Grey Forecasting* [JJFHHCLD14], ARIMA (Autoregressive Integrated Moving Average) [CMRB14] ou les réseaux de neurones comme le propose Leandro Fontoura dans sa thèse [Fon15].

Enfin, un apprentissage pourrait être mise en oeuvre à différents niveaux :

— au niveau des reconfigurations à mettre en place. Il s'agirait d'apprendre les meilleures associations entre événements détectés et reconfigurations à déclencher ; ou à anticiper les enchainements de suite d'événements et de reconfigurations.
— au niveau de l'utilisation des ressources et des bonnes combinaisons entre type d'application ; infrastructure et leviers verts utilisables.

5.3.5 Outil autonomique

Enfin, le dernier verrou à lever concerne l'outil autonomique à utiliser. En effet, c'est la brique logicielle qui sera la plus compliquée à mettre en oeuvre. Les outils gestionnaires autonomiques sont souvent complexes et implémentent tout ou une sous partie des propriétés *self-**. On trouvera dans la thèse de Rémi Sharrock [Sha10] un état de l'art des outils auquel on pourrait rajouter Frameself qui n'existait pas encore à l'époque [BAM13]. La difficulté sera ici de trouver l'outil le plus modulaire possible permettant d'intégrer notre propre formalisme d'expression de style architectural et de reconfigurations. Souvent les outils existants sont conçus pour rendre autonome chaque application qu'ils déploient et ne fournissent pas la possibilité de s'interagir avec un module de décision global et d'appliquer des reconfigurations sur plusieurs applications.

5.3.6 Reconfigurations matérielles

Les outils autonomiques sont prévus pour rendre les applications autonomes. Or, nous avons montré dans le chapitre précédent que si l'on considérait le réseau comme une application on pourrait économiser de l'énergie en allumant/éteignant des liens. Il serait donc nécessaire d'intégrer la modélisation de la plate-forme physique et des reconfigurations associées pour l'infrastructure réseau dans l'outil autonomique. Une implémentation des décisions matérielles et de leurs reconfigurations avec les SDN (Software Defined Network) permettrait une mise en oeuvre. Sur ce point, nous avons participé au dépôt d'un projet de recherche afin investiguer cette étude avec des équipes ayant des compétences réseaux complémentaires des nôtres. De plus, une collaboration avec une doctorante tunisienne se met en place sur cette thématique.

5.4 Conclusion

Nous avons proposé dans ce chapitre un framework de centre de décision autonome tel que nous pensons qu'il serait souhaitable et optimal de le construire. Une preuve de concept a été réalisée au cours du projet ANR SOP. La mise en oeuvre réelle relèverait d'un transfert industriel. Nous avons listé plusieurs verrous restant à lever : certains sont à lever par les constructeurs de machines, d'autres pourraient être levés en intégrant des travaux de recherche des domaines concernés (monitoring, prédiction). Les verrous que nous souhaiterions lever sont : la sémantisation des données et les reconfigurations matérielles à l'aide de SDN.

Chapitre 6

Conclusion et projet de recherche

6.1 Synthèse

Nous arrivons à la fin de ce mémoire, dix ans de carrière d'enseignant chercheur résumés en plus d'une centaine de pages ! J'ai présenté le résultat de participations actives à des projets de recherche nationaux, internationaux, de collaborations avec la communauté scientifique, d'encadrement scientifique : doctorants, post-doctorants. Ce travail est le résultat de coopérations et de rencontres scientifiquement enrichissantes.

J'ai obtenu ma thèse en 2004 durant laquelle j'ai étudié le placement d'applications sur cluster ou grilles de calcul en tenant compte de la qualité de service. Depuis, mes activités de recherche ont eu comme objectif de résoudre la problématique de gestion des ressources avec un objectif *vert* : celui d'économiser de l'énergie et diminuer l'impact écologique de l'*IT*.
Pour cela, j'ai investigué des actions système en considérant le système dans son ensemble, des actions au niveau du middleware en proposant différentes heuristiques de placement vertes et enfin en exploitant les principes de l'administration autonomique afin d'intégrer des reconfigurations autonomiques des applications intégrant des problématiques vertes.
J'ai exposé ce qui selon moi pourrait être un centre de décision autonome et vert. Je propose de le faire en reliant deux domaines de recherche : celui de l'optimisation et celui de l'autonomic computing.

Ces travaux se sont déroulés dans un contexte technologique qui n'a eu de cesse d'évoluer ; ainsi pendant ma thèse j'ai étudié des infrastructures de type cluster (le Beowulf

cluster date de 1994) ou grille (1998). A partir de 2005, les processeurs sont devenus multi-coeurs et en 2010 les infrastructures Cloud sont apparues. Tout ceci a nécessité d'adapter nos modèles des systèmes de calcul et nous a amenés à proposer des heuristiques plus complexes. Toutes ces évolutions sont aussi à l'origine des problématiques de consommation énergétique engendrées ! Les systèmes ont évolués, sont devenus plus performants, plus gourmands en énergie et aussi plus complexes entraînant les problématiques que nous avons traitées dans ce mémoire.

De manière générale, pour économiser de l'énergie, les approches matérielles (les avancées technologiques des constructeurs) doivent être complétées par des approches logicielles ; ainsi il y a plusieurs approches :

— *Mettre en place un refroidissement efficace.* Il existe plusieurs systèmes de refoidissement dans les datacenters. Des études ont par ailleurs montré qu'en tolérant une légère hausse des températures, il était possible d'économiser de l'énergie. Google a construit un datacenter écologique en Finlande en se basant sur la production d'énergie hydroélectrique et en utilisant les techniques de *Free Cooling* pour le refroidissement. Ces datacenters sont exploités à des températures pouvant atteindre 35 degrés [fre].
Nous proposons dans ce mémoire une approche logicielle intégrant les coûts de refroidissement : nous avons proposé des heuristiques de placement des applications intégrant des problématiques de recirculation de chaleur.

— *Réduire les pertes engendrées lors de la distribution de l'électricité.* Les PSU (Power Supply Unit) sont limités à une efficacité de 80% ; ainsi des efforts sont en cours pour les supprimer et utiliser du courant DC. Ceci a deux avantages : un gain de place dans les datacenters et un gain d'efficacité énergétique grâce à la réduction du refroidissement.
Nous n'avons pas encore travaillé sur cette piste de recherche mais nous prévoyons de l'étudier.

— *Choisir des processeurs avec de hauts rapports Flops par Watt.* C'est une possibilité que les propriétaires de datacenters ont : choisir du matériel performant et énergétiquement efficace. Nous avons choisi une approche complémentaire en proposant la construction d'infrastructures à consommation proportionnelle en exploitant des architectures hétérogènes : ces infrastructures bénéficient des avantages des architectures performantes lorsque la charge est importante tout en étant économes lorsque la charge est faible. Nous avons donc transposé le concept big.LITTLE aux infrastructures HPC ou cloud.

— *Utiliser des outils energy aware.* Nous avons pour cela proposé des algorithmes

de placement résolvant des optimisations multi-critères. Les heuristiques proposées tiennent compte du critère énergétique. Les solutions que nous avons proposées ont été évaluées dans le cadre de projets recherche nationaux et internationaux démontrant leur efficacité. Grâce à des algorithmes de décision efficaces, nous avons démontré qu'il était possible d'obtenir des compromis entre l'énergie dépensée et la dégradation de performance. Nous avons proposé une heuristique *fuzzy-based priority* qui est une alternative intuitive pour décrire les compromis dans des problèmes d'optimisation multi-critères et dont les solutions se trouvent sur la frontière de pareto.
Nous avons également montré qu'il était possible en considérant le système dans son ensemble d'activer les leviers verts bas niveau appropriés en détectant des phases dans l'exécution de l'ensemble des applications.

— *Optimiser les applications.* Cette piste peut se mettre en oeuvre en modifiant les codes des applications, en les re-développant avec l'optique quelles consomment moins d'énergie. Ainsi, par exemple, en optimisant les accès mémoire d'une application on peut à la fois augmenter ses performances et diminuer la consommation d'énergie.
Nous avons choisi une autre approche : nous proposons d'optimiser les applications en appliquant les propriétés *self-** des systèmes autonomiques. Ceci, ne peut bien sûr s'appliquer qu'aux applications adaptables et exploitent leurs caractéristiques d'élasticité et de dynamicité. Nous considérons les applications comme des systèmes dynamiques autonomes actifs dans leurs adaptations et mettant en oeuvre des reconfigurations optimisant à la fois leur performance mais aussi leur consommation d'énergie.

Nous avons proposé une approche complète avec un centre de décision autonome intégrant plusieurs briques logicielles dont un placeur et un gestionnaire autonomique. Chaque brique intégrant les éléments de recherche sur lesquels nous avons travaillé.

6.2 Projet de recherche

Les perspectives de recherche du *Green Computing* sont nombreuses. Nous allons évoquer ici les pistes de recherche que nous souhaitons investiguer à court et moyen terme.

6.2.1 Projet de recherche à court terme

L'opérationnalisation du centre de décision vert et autonome. Nous avons cité dans le chapitre précédent les verrous restant à lever avant d'obtenir une solution opération-

nelle. Nous allons détailler les perspectives à court terme sur lesquelles nous envisageons de travailler.

— Prise en compte de différents types d'événements. Les événements à prendre en compte lors de reconfigurations par un centre de décision vert sont variés ; afin de généraliser l'approche, il parait intéressant de prendre en compte des événements liés à l'approvisionnement en électricité indiquant un problème d'alimentation de machines ou d'une partie de l'infrastructure. Actuellement, la résilience dans les centres de calcul est gérée en fournissant une redondance dans le circuit d'alimentation du centre de calcul. Ceci engendre un coût et des pertes électriques que l'on pourrait éviter ou diminuer en utilisant le centre de décision pour détecter les défaillances. Sur réception de ces événements la politique de reconfiguration devra mettre en oeuvre différents leviers pour éviter un arrêt des applications s'exécutant sur la partie de l'infrastructure impliquée dans l'événement. Il s'agira d'optimiser les exécutions en cours : choisir les applications critiques à regrouper et à migrer afin qu'elles continuent à fonctionner ; choisir les applications à éteindre afin d'économiser l'électricité au profit des applications critiques.

— Sémantisation. Les reconfigurations seraient aussi plus efficaces en ajoutant du sens aux événements comme nous l'avons expliquer au chapitre précédent. La sémantisation peut être utile sur différents points : pour identifier les symptômes, pour fournir une base de raisonnement (aide à la décision ou diagnostic). La sémantisation se rajoute dans la base de connaissances de la boucle MAPE-K en complétant la formalisation du domaine applicatif et de l'infrastructure. Nous avons débuté une collaboration avec l'équipe MELODI sur ce point lors d'une réponse à appel à projet. Toute la difficulté consistera à permettre un raisonnement sur les événements reçus (détection d'événements périmés, agrégation d'événements...) et déclencher des reconfigurations adaptées ; pour cela, nous utiliserons des ontologies.

— Optimisation. Nous envisageons de poursuivre nos travaux sur les algorithmes d'optimisation multi-critères. Nous sommes notamment en cours d'étude d'une intégration dans une heuristique de placement d'une prédiction des machines à allumer dans le futur en utilisant un algorithme génétique. C'est une approche innovante visant à utiliser les GA pour trouver la meilleure combinaison de variables représentant l'état d'un système et à utiliser cette combinaison dans l'itération suivante de placement. L'idée consiste à prédire le nombre de machines physiques qui seront nécessaires à garder allumées dans le futur puis à optimiser le placement sur cette sous-partie de l'infrastructure.
Nous souhaitons également prendre en compte le critère écologique dans les problèmes d'optimisation afin d'intégrer des sources d'énergie renouvelables (détaillé au paragraphe suivant).
Nos algorithmes de décision devront aussi intégrer les coûts des reconfigurations

et choisir la meilleure reconfiguration possible en tenant compte : du coût de la transition d'une configuration vers une autre, des gains obtenus sur les différentes métriques (énergie, performance) et éventuellement choisir une configuration qui n'est pas la meilleure dans l'absolu mais la meilleure relativement aux surcoûts engendrés et aux bénéfices attendus. De même, nous avons jusqu'à présent travaillé en considérant les propriétés *self-** indépendamment les unes des autres. Or, il peut parfois être nécessaire de trouver un compromis entre les propriétés *self-healing* et *self-optimizing*. L'optimisation doit en tenir compte notamment en s'aidant de la sémantisation associée aux événements reçus.

— Reconfigurations matérielles. Nous avons mentionné dans le chapitre précédent que les SDN (Software Defined Network) permettraient une implémentation des décisions matérielles et des reconfigurations de l'infrastructure réseau. SDN sépare le plan de contrôle du plan des données des équipements réseau. Ainsi, un routeur est transformé en un dispositif d'expédition simple qui applique des règles envoyées par un contrôleur distant en utilisant un protocole normalisé. Cette approche simple permet aux administrateurs de réseau d'obtenir un meilleur contrôle sur le trafic dans leur réseau. Ainsi, le contrôleur pourrait résoudre le problème d'optimisation proposé en utilisant des matrices de trafic collectées, calculer la solution de routage amenant le meilleur compromis de QoS/énergie puis mettre en oeuvre la solution trouvée en mettant à jour les règles d'expédition des équipements réseaux et en éteignant les interfaces inutilisées. Nous proposons de travailler à deux niveaux :
  — Placement des contrôleurs de SDN. Îl s'agit d'optimiser l'infrastructure de contrôle en choisissant le nombre de contrôleurs et leur placement dans l'infrastructure afin de réduire la latence des communications entre le contrôleur et les noeuds.
  — Allocation des ressources. Il s'agit d'optimiser l'allocation des ressources au niveau du contrôleur pour mettre en place un routage efficace en énergie et prendre les décisions d'allumage et extinction de liens à partir des données de trafic collectées et de la qualité de service souhaitée.

— Coordination de gestionnaires autonomiques. Nous nous sommes focalisés dans le chapitre précédent sur la construction d'un centre de décision autonome. Toutefois, pour gérer le passage à l'échelle des systèmes, on aura probablement des centres de décisions distribués nécessitant une coordination et/ou une synchronisation des décisions afin de conserver une cohérence des reconfigurations. Pour cela, il faudra étudier deux verrous scientifiques :
  — Gestion des événements concurrents. Dans les systèmes à large échelle, nous serons confrontés au problème de la réception d'événements concurrents. La sémantisation nous aidera à prioriser les événéments mais il est aussi envisageable de compléter l'approche en utilisant le CEP (Complex Event Processing) pour adresser le problème de gestion des événements complexes. Nous pourrons ainsi

définir des clauses d'agrégation des événements observés par les sondes et ainsi définir des règles pour exécuter une seule reconfiguration plutôt qu'une série de reconfigurations.

— Décision distribuée. Le passage à l'échelle nécessite de construire le module de décision décentralisé en étudiant les stratégies de coordination. Pour cela, il sera intéressant d'utiliser la théorie des jeux.

A plus long terme, nous envisageons d'étudier deux perspectives de recherche détaillées dans les paragraphes qui suivent : l'intégration de sources d'énergie renouvelables et la gestion de ville intelligente.

6.2.2 Projet de recherche à plus long terme

Intégrer des sources d'énergie renouvelables. Nous sommes convaincus que pour obtenir des datacenters verts il faut intégrer des sources d'énergie vertes. Dans son rapport [how] souligne que même si l'efficacité énergétique augmente, la croissance exponentielle des clouds oblige à considérer plusieurs sources d'énergie et notamment des sources d'énergies renouvelables. En effet, sans cela, on ne peut pas vraiment parler de centre de décision "vert" si la principale source d'énergie est le charbon ou le nucléaire. On ne pourra parler de green IT que si on associe efficacité énergétique et énergie renouvelable. Il n'est cependant pas réaliste d'imaginer que le datacenter pourrait s'auto-alimenter.
"De l'énergétique au thermique, le Green IT est il réellement vert ?" était d'ailleurs le thème des GreenDays@Toulouse [1] organisées par l'équipe en mars 2015 à l'IRIT. Dans [BDFG+15], les auteurs présentent le projet EPOC qui vise à optimiser la consommation d'énergie d'un datacenter connecté au réseau électrique et bénéficiant de sources d'énergie renouvelables.

Nous aimerions donc travailler sur des modèles intégrant plusieurs sources d'énergie : solaire, hydraulique, géothermie, nucléaire. L'objectif sera de se rapprocher d'un centre de calcul avec zéro émission de carbone. Les problématiques abordées seront :

— Modélisation des sources d'énergie. Il s'agira de modéliser les différentes sources d'alimentation en électricité en termes de profil de disponibilité, de puissance disponible, de leviers utilisables (avec les coûts et les risques associés). Nous ne prévoyons pas de travailler à la proposition de tels modèles mais plutôt d'utiliser des modèles de la communauté travaillant sur cette thématique. Les modèles des sources d'énergie associés à des méthodes d'apprentissage ou de prédiction pourront être utilisés pour planifier la disponibilité des sources d'énergie.

— Optimisation de l'alimentation du centre de calcul. A partir des modèles précédents, il s'agira de choisir les sources pour l'alimentation des serveurs, du système de re-

1. homepage : perso.ens-lyon.fr/laurent.lefevre/greendaystoulouse/

froidissement.

— Solution hybride optimale adaptative. Une solution hybride basée sur le réseau électrique classique et des sources d'énergie renouvelables peut être proposée. Se posera la question du choix des sources d'énergie et surtout de l'optimisation des tâches pour exploiter au maximum l'énergie lorsqu'elle est disponible. En effet, le stockage de l'énergie est un défi à relever pour exploiter au mieux les énergies renouvelables afin de résoudre le problème d'intermittence des sources d'énergie. Des heuristiques de placement et des algorithmes de planification des tâches seront donc nécessaires. Le verrou scientifique à lever sera l'adaptation de la gestion des ressources à la production d'électricité. En effet, le problème d'optimisation des tâches se complexifie en intégrant une dimension supplémentaire celle de l'alimentation électrique. Il faudra envisager des politiques d'allocation permettant de déplacer ou suspendre des tâches en fonction de la disponibilité des ressources d'énergie ou au contraire, exploiter un surplus d'énergie disponible en lançant des tâches qui n'étaient à l'origine pas planifiées à ce moment là. C'est donc une généralisation des problèmes d'optimisation présentés dans ce mémoire en proposant un ordonnancement et plus uniquement un placement. De plus, de telles approches nécessiteront l'intégration de critères supplémentaires : celui du coût financier de la source d'énergie utilisée, celui du coût d'émission de carbone. Il faudra donc identifier les critères d'optimisation pertinents. Techniquement, il sera nécessaire de faire évoluer les simulateurs de centres de calcul afin notamment d'intégrer les modèles des sources d'énergie. Par ailleurs, le centre de décision sera utile pour détecter des événements de puissance (défaillance d'une source d'énergie, pic de production, sous-production ...) et déclencher des reconfigurations des applications, de nouvelles décisions de placement ou adapter la source d'énergie au besoin applicatif et éviter le sur-approvisionnement en ressources.

Ces sources d'énergie vertes malgré le fait que l'on ne maîtrise pas leur disponibilité ne sont pas négligeables. On peut d'ailleurs espérer des innovations technologiques qui les rendront encore plus efficaces. Un procédé innovant a été par exemple testé pour continuer à produire de l'électricité la nuit avec des panneaux solaires grâce à la chaleur accumulée pendant la journée [BE].
La mise en oeuvre de tels datacenters peut s'appuyer sur l'implication des utilisateurs à qui l'on peut proposer l'utilisation des services lorsque l'énergie est disponible.

La maison intelligente, la ville intelligente. Cette perspective est la suite et la transposition de la précédente au domaine de la maison ou à l'échelle d'une ville et non plus à l'échelle d'un datacenter. L'ADEME a calculé le coût électrique de nos actions digitales, à titre d'exemple, une heure d'échange de mail (en considérant en moyenne une pièce jointe par mail) à l'échelle d'Internet équivaut à 4000 aller-retours Paris-New York en avion. Pour les usagers, les moyens de calcul et stockage sont tellement démocratisés qu'ils n'ont pas

conscience des conséquences écologiques des actions du quotidien. Le volume des données en circulation double tous les deux ans. Or, ces données sont hébergées dans le Cloud entraînant une explosion des données stockées, c'est ce que l'on appelle le *Big Data*. Les usagers se voient proposer des services de stockage de données à des prix battant toute concurrence et ne font même plus l'effort de supprimer des données obsolètes auxquelles ils n'accèderont plus jamais !

Dans son livre "La troisième révolution industrielle" Jeremy Rifkin [JR] a observé que les grandes révolutions économiques de l'histoire se sont produites lorsque de nouvelles technologies de la communication sont apparues en même temps que de nouvelles sources d'énergie. Selon lui "la Troisième révolution industrielle sera le fruit d'une synergie détonante entre les énergies renouvelables et les technologies Internet, qui modifiera les modes de distribution de l'énergie au XXIème siècle. Dans l'ère à venir, des centaines de millions de personnes produiront leur propre énergie verte à la maison, au bureau et à l'usine, et elles se la partageront via un système d'Internet de l'énergie distribuée." Jeremy Rifkin propose de s'appuyer sur l'usager en proposant une révolution énergétique où les utilisateurs peuvent être à la fois consommateurs et producteurs.

Concrètement, ce que nous souhaiterions étudier c'est le cas d'une maison ou d'une ville intelligente profitant dans le meilleur des cas de plusieurs sources d'énergie. En intégrant les technologies du domaine de l'IoT (l'Internet des Objets) : les capteurs et actionneurs, les technologies de réseaux assurant l'interopérabilité puis en collectant les données nous souhaiterions proposer une gestion intelligente des ressources. En analysant les données collectées, en mettant en place des techniques d'apprentissage et de planification nous pourrions permettre un contrôle optimisé des appareils et des services interconnectés dans la maison ou la ville intelligente. D'autre part, il serait possible de proposer à l'utilisateur des adaptations de ses usages en lui permettant de piloter ses consommations.

Différentes problématiques recherche seront soulevées dans le contexte de la maison, du bâtiment et de la ville intelligente :

— Changement d'usage : modèle producteur-consommateur. On peut imaginer que le réseau de distribution de l'électricité sera de plus en plus intelligent permettant un partage de l'électricité produite par les différents utilisateurs. Les différents bâtiments auront différentes sources de production d'électricité pouvant être mises à disposition des autres utilisateurs en fonction des besoins. On peut imaginer au niveau d'un quartier un centre de décision permettant d'optimiser les consommations par exemple en ordonnançant les différentes tâches (au niveau d'une habitation : lessive, lave-vaisselle....) en fonction des sources d'énergie disponibles. Les réseaux intelligents grâce aux technologies par exemple de compteur intelligent permettront d'agir sur la demande et permettront d'adapter, en partie, la consommation aux capacités instantanées de production, notamment en décalant certaines consommations en dehors des heures de pointe. L'adéquation d'une partie de la consommation (industrielle et domestique) à la production disponible, permettra ainsi de diminuer les pics de consommation et donc d'exploiter au mieux la production dans une zone

géographique donnée.

— Datacenters distribués chez les particuliers. Nous avons vu dans ce mémoire que la chaleur produite par les centres de calcul est gaspillée et engendre des coûts de refroidissement importants. On peut imaginer la généralisation des prototypes de "radiateurs-calculateurs" de la société Qarnot-Computing [2] qui propose un nouveau mode de chauffage gratuit et non-polluant en distribuant les centres de calcul dans les bâtiments. Les centres de calcul sont ainsi déportés chez les particuliers. Les radiateurs sont composés de processeurs et exécutent plus ou moins de calcul en fonction de la température souhaitée.
Une généralisation de ce fonctionnement nécessitera des centres de décision distribués permettant de résoudre l'optimisation des choix des calculs envoyés sur les différents calculateurs.

— Optimisation de l'infrastructure. Les scénarios de maison ou de ville intelligente reposent sur des objets connectés. On parle d'"Internet des Objets". La multiplication de ces objets connectés hétérogènes nécessitera une puissance de calcul de plus en plus importante pour le traitement des données, et le recours à des services distants de stockage et d'analyse de contenus. Il est alors nécessaire d'intégrer ces services dans le Cloud et d'optimiser l'infrastructure de service. De plus, la ville intelligente devient une mise en application de scénarios d'économie d'énergie en pilotant les différents équipements connectés. Le centre de décision peut être utilisé pour détecter des anomalies, identifier des dysfonctionnements et mettre en oeuvre les reconfigurations nécessaires.

2. http ://www.qarnot-computing.com/

Annexe A

Publications

Publications

[1] D. Borgetto, R. Chakode, B. Depardon, C. Eichler, J.-M. Garcia, H. Hbaieb, T. Monteil, E. Pelorce, A. Rachdi, A. Al Sheikh, and P. Stolf, "Hybrid approach for energy aware management of multi-cloud architecture integrating user machines," *Journal of Grid Computing, Green Cloud Computing*, 2015.

[2] V. Villebonnet, G. Da Costa, L. Lefèvre, J.-M. Pierson, and P. Stolf, "Big, Medium, Little : Reaching Energy Proportionality with Heterogeneous Computing Scheduler," *Parallel Processing Letters*, 2015.

[3] L. Fontoura Cupertino, G. Da Costa, A. Oleksiak, W. PiaTek, J.-M. Pierson, J. Salom, L. Siso, P. Stolf, H. Sun, and T. Zilio, "Energy-Efficient, Thermal-Aware Modeling and Simulation of Datacenters : The CoolEmAll Approach and Evaluation Results," *Ad Hoc Networks Journal*, vol. 25, no. B, pp. 535–553, février 2015.

[4] G. L. Tsafack Chetsa, L. Lefèvre, and P. Stolf, "A Three Step Blind Approach for Improving HPC Systems' Energy Performance," *Concurrency and Computation :Practice and Experience*, vol. 10.1002/cpe.3312, pp. 1–18, juillet 2014.

[5] H. Sun, P. Stolf, J.-M. Pierson, and G. Da Costa, "Energy-efficient and thermal-aware resource management for heterogeneous datacenters," *Sustainable Computing : Informatics and Systems*, vol. 10.1016/j.suscom.2014.08.005, pp. 1–15, août 2014.

[6] C. Eichler, T. Monteil, P. Stolf, A. Grieco, and K. Drira, "Enhanced Graph Rewriting Systems for Complex Software Domain." *Software and Systems Modeling*, vol. 10.1007/s10270-014-0433-1, pp. 1–21, septembre 2014.

[7] C. Eichler, G. Gharbi, T. Monteil, P. Stolf, and N. Guermouche, "Self-management of Machine-to-Machine communications : a multi-models approach," *International Journal of Autonomous and Adaptive Communications Systems*, 2014.

[8] G. L. Tsafack Chetsa, L. Lefèvre, J.-M. Pierson, P. Stolf, and G. Da Costa, "Exploiting performance counters to predict and improve energy performance of HPC systems," *Future Generation Computer Systems*, vol. 10.1016/j.future.2013.07.010, pp. 287–298, juillet 2014.

[9] M. Diouri, G. L. Tsafack Chetsa, O. Gluck, L. Lefèvre, J.-M. Pierson, P. Stolf, and G. Da Costa, "Energy efficiency in HPC : with or without knowledge on applications

and services," *International Journal of High Performance Computing Applications*, vol. doi :10.1177/1094342013497990, pp. 232–243, août 2013.

[10] R. Sharrock, T. Monteil, P. Stolf, D. Hagimont, and L. Broto, "Non-intrusive autonomic approach with self-management policies applied to legacy infrastructures for performance improvements." *International Journal of Adaptive, Resilient and Autonomic Systems*, vol. 2, no. 2, pp. 58–76, 2011.

[11] N. Hernandez, J. Mothe, B. J. V. Ralalason, A. B. Ramamonjisoa, and P. Stolf, "A Model to Represent the Facets of Learning Objects," *Interdisciplinary Journal of E-Learning and Learning Objects*, vol. 4, pp. 65–82, janvier 2008.

[12] L. Lefèvre, S. Varrette, F. Pinel, P. Bouvry, G. L. Tsafack Chetsa, G. Da Costa, P. Stolf, J.-M. Pierson, and E. Jeannot, "Energy Efficiency and High Performance Computing," in *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*, J.-M. Pierson, Ed., John Wiley and Sons, mai 2015, ch. 7, pp. 197–224, iSBN : 978-1-118-86463-0.

[13] A. Gazo Cervero, M. Chincoli, L. Dittmann, A. Fischer, A. E. Garcia, L. Lefèvre, G. Lovász, H. De Meer, P. Monti, A.-C. Orgerie, L.-F. Pau, C. Phillips, P. Stolf, L. Valcarenghi, T. Monteil, R. Sharrock, T. Trinh, and S. Ricciardi, "Green Wired Networks," in *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*, J.-M. Pierson, Ed., John Wiley and Sons, mai 2015, ch. 3, pp. 41–80.

[14] D. Hagimont, P. Stolf, L. Broto, and N. Depalma, "Component-based autonomic management for legacy software," in *Autonomic Computing and Networking*, ser. ISBN : 978-0-387-89827-8, M. Denko, L. Yang, and Y. Zhang, Eds. Springer, 2009, pp. 83–104.

[15] N. Hernandez, J. Mothe, A. B. Ramamonjisoa, B. J. V. Ralalason, and P. Stolf, "Indexation multi-facettes des ressources pédagogiques pour faciliter leur ré-utilisation," *Revue des Nouvelles Technologies de l'Information, Modelisation des Connaissances*, vol. 12, no. 1, pp. 229–254, 2008.

[16] I. Khlif, M. Hadj Kacem, P. Stolf, and A. Hadj Kacem, "Software architectures : multi-scale refinement (regular paper)," in *International Conference on Software Engineering Research, Management and Applications (SERA), Hammamet, Tunisie, 13/05/2015-15/05/2015*, IEEE Computer Society, 2015.

[17] G. L. Tsafack Chetsa, L. Lefèvre, J.-M. Pierson, P. Stolf, and G. Da Costa, "Application-Agnostic Framework for Improving the Energy Efficiency of Multiple HPC Subsystems ," in *Parallel, Distributed, and Network-Based Processing (PDP), Turku, Finland, 04/03/2015-06/03/2015*,IEEE, mars 2015, pp. 62–69.

[18] V. Villebonnet, G. Da Costa, L. Lefèvre, J.-M. Pierson, and P. 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*, IEEE, décembre 2014, pp. 703–710.

[19] D. Borgetto and P. Stolf, "An Energy Efficient Approach to Virtual Machines Management in Cloud Computing. (regular paper)," in *International Conference on Cloud Networking, Luxembourg, 08/10/2014-10/10/2014*, IEEE, 2014, pp. 229–235.

[20] W. Akio Higashino, C. Eichler, M. A. M. Capretz, T. Monteil, M. B. F. De Toledo, and P. Stolf, "From Inception to Execution : Query Management for Complex Event Processing as a Service (short paper)," in *IEEE International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE), Parme, Italie, 23/06/14-25/06/14*, IEEE Computer Society, pp. 107-109, juin 2014.

[21] H. Sun, P. Stolf, J.-M. Pierson, and G. 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, mai 2014.

[22] C. Eichler, G. Gharbi, T. Monteil, N. Guermouche, and P. Stolf, "Graph-based formalism for Machine-to-Machine self-managed communications (regular paper)," in *IEEE International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE), Hammamet (Tunisie), 17/06/2013-22/06/2013*, IEEE, juin 2014, pp. 74–79.

[23] G. L. Tsafack Chetsa, L. Lefèvre, J.-M. Pierson, P. Stolf, and G. 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, août 2013, pp. 118–125.

[24] R. Sharrock, T. Monteil, P. Stolf, and O. Brun, "Autonomic computing to manage green Core networks with Quality of Service (regular paper)," in *Energy Efficiency in Large Scale Distributed Systems conference (EE-LSDS), Vienna, 22/04/2013-24/04/2013*, Springer, 2013, pp. 248–263.

[25] G. L. Tsafack Chetsa, L. Lefèvre, and P. Stolf, "A Three Step Blind Approach for Improving HPC Systems' Energy Performance (regular paper)," in *Energy Efficiency in Large Scale Distributed Systems conference (EE-LSDS), Vienna, 22/04/2013-24/04/2013*, Springer, 2013, pp. 168–181.

[26] G. L. Tsafack Chetsa, L. Lefèvre, J.-M. Pierson, P. Stolf, and G. 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, octobre 2012, pp. 132–138.

[27] G. L. Tsafack, L. Lefèvre, J.-M. Pierson, P. Stolf, and G. Da Costa, "A runtime framework for energy efficient hpc systems without a priori knowledge of applications," in *ICPADS 2012 : 18th International Conference on Parallel and Distributed Systems*, (Singapore, Singapore), pp. 660–667, IEEE, Dec 2012.

[28] G. L. Tsafack, L. Lefèvre, J.-M. Pierson, P. Stolf, and G. Da Costa, "Dna-inspired scheme for building the energy profile of hpc systems," in *Proceedings of the First international conference on Energy Efficient Data Centers*, E2DC'12, (Berlin, Heidelberg), pp. 141–152, Springer-Verlag, 2012.

[29] R. Sharrock, P. Stolf, T. Monteil, and T. Guerout, "Internal self-protecting for consistency and stability in an autonomic manager (regular paper)," in *IEEE International Symposium on Network Computing and Applications (IEEE NCA), Toulouse, 21/10/2011-23/10/2011*, ser. ISBN : 978-0-7695-4550-9, IEEE Computer Society, 2011, pp. 44–49.

[30] A. Ortiz, F. Thiébolt, P. Stolf, G. Da Costa, and A. Sayah, "Virtual Machine Migration : A Comparative Study of Storage Viewpoints (regular paper)," in *International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Ajaccio, Corsica, France, 12/04/2011-15/04/2011*, avril 2011.

[31] D. Borgetto, P. Stolf, G. Da Costa, and J.-M. Pierson, "Energy Aware Autonomic Manager (poster)," in *ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy), Passau, Germany, 13/04/2010-15/04/2010*, avril 2010.

[32] M. Toure, P. Stolf, D. Hagimont, and L. Broto, "Large scale deployment (regular paper)," in *International Conference on Autonomic and Autonomous Systems (ICAS), Cancun, Mexico, 07/03/2010-13/03/2010*, IEEE, 2010.

[33] R. Sharrock, P. Stolf, and T. Monteil, "Extending TUNe for autonomous management of QoS at application and network levels (short paper)," in *International Conference on Networks (ICN), Menuires, France, 11/04/2010-16/04/2010*,IEEE, 2010.

[34] R. Sharrock, F. Khalil, T. Monteil, H. Aubert, F. Coccetti, P. Stolf, L. Broto, and R. Plana, "Deployment and management of large planar reflectarray antennas simulation on grid (regular paper)," in *Challenges of Large Applications in Distributed Environments, Workshop in conjunction with HPDC 2009 (CLADE), Munich, 09/06/2009-10/06/2009*, ACM, 2009, pp. 17–26.

[35] L. Broto, D. Hagimont, P. Stolf, N. Depalma, and S. Temate, "Autonomic management policy specification in Tune," in *Annual ACM Symposium on Applied Computing (SAC), Fortaleza, Ceará, Brazil, 16/03/2008-20/03/2008*, ACM, mars 2008, pp. 1658–1663.

[36] M. Toure, G. Berhe, P. Stolf, L. Broto, N. Depalma, and D. Hagimont, "Autonomic Management for Grid Applications," in *Euromicro International Conference on Parallel, Distributed and network-based Processing, Toulouse, 13/02/2008-15/02/2008*, IEEE, février 2008, pp. 79–86.

[37] N. Hernandez, J. Mothe, B. J. V. Ralalason, A. B. Ramamonjisoa, and P. Stolf, "Multifacet indexing for learning objects reuse," in *Computer Science Information Techno-*

logy Education, Pointe aux Sables, 16/11/2007-18/11/2007, Institute for Scientific Information (ISI), novembre 2007, pp. 309–322.

[38] N. Hernandez, J. Mothe, B. J. V. Ralalason, and P. Stolf, "Modèle de représentation sémantique des documents électroniques pour leur réutilisabilité dans l ?apprentissage en ligne," in *Colloque International sur le Document Électronique (CIDE), Fribourg (Suisse), 18/09/2006-20/09/2006*, EUROPIA, 2006, pp. 181–198.

[39] C. Eichler, I. Bouassida, K. Drira, T. Monteil, and P. Stolf, "Caractérisation de la reconfiguration dynamique des architectures logicielles par les grammaires de graphe (regular paper)," in *Conférence Francophone sur les Architectures Logicielles (CAL), Montpellier, France, 30/05/2012-31/05/2012*, ACM, mai 2012, pp. 1–11.

[40] R. Sharrock, P. Stolf, and T. Monteil, "How to enhance the grid accessibility for non-expert users using autonomic computing. (regular paper)," in *Grid'5000 Spring School Conference, Lille, 06/04/2010-09/04/2010*, CNRS EDITIONS, 2010, pp. 1–6.

[41] L. Broto, P. Stolf, J.-P. Bahsoun, D. Hagimont, and N. Depalma, "Spécification de politiques d'administration autonome avec Tune," in *Conférence Française sur les Systèmes d'Exploitation (CFSE), Fribourg, Suisse, 11/02/2008-13/02/2008*, Chapitre Français de l'ACM SIGOPS, février 2008.

Bibliographie

[ABP11] U. Ayesta, O. Brun, and B.J. Prabhu. Price of anarchy in non-cooperative load-balancing games. *Performance Evaluation*, 68 :1312–1332, 2011.

[acp] Acpi specification. `www.acpi.info/spec.htm`.

[ADG98] Robert Allen, Rémi Douence, and David Garlan. Specifying and analyzing dynamic software architectures. In *Fundamental Approaches to Software Engineering*, volume 1382 of *Lecture Notes in Computer Science*, pages 21–37. Springer Berlin Heidelberg, 1998.

[AGH04] I. Assayad, A. Girault, and Kalla H. A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, page 347, 2004.

[AHEAMC+14] Wilson Akio Higashino, Cédric Eichler, Miriam A. M. Capretz, Thierry Monteil, Maria Beatriz F. De Toledo, and Patricia Stolf. From Inception to Execution : Query Management for Complex Event Processing as a Service (short paper). In *IEEE International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE), Parme, Italie, 23/06/14-25/06/14*, http ://www.computer.org, juin 2014. IEEE Computer Society.

[AHTH14] S. Akiyama, T. Hirofuchi, R. Takano, and S. Honiden. Fast Live Migration with Small IO Performance Penalty by Exploiting SAN in Parallel. In *IEEE 7th International Conference on Cloud Computing (CLOUD)*, June 2014.

[ALW10] L.H. Andrew, M. Lin, and A. Wierman. Optimality, fairness, and robustness in speed scaling designs. In *Proceedings of ACM SIGMETRICS*, pages 37–48, 2010.

[Awo06] Steve Awodey. *Category Theory*, volume 49 of *Oxford Logic Guides*. Oxford University Press, 2006.

[BAB12] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.*, 28(5) :755–768, May 2012.

[BAM13] M. Ben Alaya and T. Monteil. FRAMESELF : An ontology-based framework for the self-management of M2M systems. *Concurrency and Computation : Practice and Experience*, 10.1002/cpe.3168, 2013.

[BBB+91] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks. Technical report, The International Journal of Supercomputer Applications, 1991.

[BBS+00] D.M. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P.W. Cook. Power-aware microarchitecture : Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6) :26–44, 2000.

[BCC+06] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Iréa Touche. Grid'5000 : A large scale and highly reconfigurable experimental grid testbed. *Int. J. High Perform. Comput. Appl.*, 20(4) :481–494, November 2006.

[BCC+13] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science*, Communications in Computer and Information Science. Springer Int. Publishing, 2013.

[BCD+15] D. Borgetto, R. Chakode, B. Depardon, C. Eichler, JM. Garcia, H. Hbaieb, T. Monteil, E. Pelorce, A. Rachdi, A. Al Sheikh, and P. Stolf. Hybrid approach for energy aware management of multi-cloud architecture integrating user machines. *Journal of Grid Computing, special issue on Green Cloud Computing (à paraître)*, 2015.

[BCDCP12] D. Borgetto, H. Casanova, G. Da Costa, and J.M. Pierson. Energy-aware service allocation. *Future Generation Computer Systems*, 28(5) :769–779, 2012.

[BCDW04] Jeremy S. Bradbury, James R. Cordy, Juergen Dingel, and Michel Wermelinger. A survey of self-management in dynamic software architecture specifications. In *ACM SIGSOFT workshop on Self-managed systems (WOSS)*, pages 28–33, New York, NY, USA, 2004. ACM.

[BCL+06] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J. Stefani. The fractal component model and its support in java. *Software : Practice and Experience*, 36(11-12) :1257—1284, 2006.

[BDF+14] M. Bertier, F. Desprez, G. Fedak, A. Lebre, A.C. Orgerie, J. Pastor, F. Quesnel, J. Rouzaud-Cornabas, and C. Tedeschi. Beyond the clouds : How should next generation utility computing infrastructures be designed ? *Cloud Computing*, pages 325–345, 2014.

[BDFG+15] Nicolas Beldiceanu, Barbara Dumas Feris, Philippe GRAVEY, Md Sabbir Hasan, Claude Jard, Thomas Ledoux, Yunbo Li, Didier Lime, Gilles Madi-Wamba, Jean-Marc Menaud, Pascal Morel, Michel Morvan, Marie-Laure Moulinard, Anne-Cécile Orgerie, Jean-Louis Pazat, Olivier Roux, and Ammar Sharaiha. The EPOC project : Energy Proportional and Opportunistic Computing system. In *International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)*, Lisbonne, Portugal, May 2015.

[BE] Vive l'énergie solaire... la nuit ! `http://www.bulletins-electroniques.com/actualites/077/77392.htm`.

[BH07] L.A. Barroso and U. Holzle. The case for energy-proportional computing. *IEEE Computer*, 40(12) :30–37, 2007.

[BHK+04] Kurt Binder, Jürgen Horbach, Walter Kob, Wolfgang Paul, and Fathollah Varnik. Molecular dynamics simulations. *Journal of Physics : Condensed Matter*, 16(5) :S429, 2004.

[BHS+08] L. Broto, D. Hagimont, P. Stolf, N. de Palma, and S. Temate. Autonomic management policy specification in tune. In *ACM Symposium on Applied Computing*, pages 1658–1663, Fortaleza, Ceara, Brazil, 2008.

[BHTV06] Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Dániel Varró. Style-Based Modeling and Refinement of Service-Oriented Architectures. *Journal of Software and Systems Modelling*, 5(2) :187–207, June 2006.

[big13] ARM big.LITTLE Technology : The Future of Mobile. *ARM White Paper*, 2013.

[BMDC+12] Damien Borgetto, Michael Maurer, Georges Da Costa, Ivona Brandic, and Jean-Marc Pierson. Energy-efficient and SLA-Aware Management of IaaS Clouds. In *ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy), Madrid, 09/05/2012-11/05/2012*. ACM DL, 2012.

[BMVG11] A. Banerjee, T. Mukherjee, G. Varsamopoulos, and S.K. Gupta. Integrating cooling awareness with thermal aware workload placement for HPC data centers. *Sustainable Computing : Informatics and Systems*, 1(2) :134–150, 2011.

[Bor13] Damien Borgetto. *Allocation et réallocation de services pour les économies d'énergie dans les clusters et les clouds.* PhD thesis, Université Toulouse 3 Paul Sabatier, 2013.

[BRC10] R. Buyya, R. Ranjan, and R.N. Calheiros. Intercloud : Utility-oriented federation of cloud computing environments for scaling of application services. In *10th Int. Conf. on Algorithms and Architectures for Parallel Processing, ICA3PP'10*, pages 13–31, 2010.

[Bro08] Laurent Broto. *Support langage et système pour l'administration autonome.* PhD thesis, Université de Toulouse, Toulouse, France, September 2008.

[BS14] Damien Borgetto and Patricia Stolf. An Energy Efficient Approach to Virtual Machines Management in Cloud Computing. (regular paper). In *International Conference on Cloud Networking, Luxembourg, 08/10/2014-10/10/2014*, pages 229–235, http ://www.ieee.org/, 2014. IEEE.

[BSB+08] L. Broto, P. Stolf, J. P. Bahsoun, D. Hagimont, and N. de Palma. Spécification de politiques d'administration autonome avec tune. In *RenPar'18 / SympA'2008 / CFSE'6*, 2008.

[BSDCP10] Damien Borgetto, Patricia Stolf, Georges Da Costa, and Jean-Marc Pierson. Energy Aware Autonomic Manager (poster). In *ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy), Passau, Germany, 13/04/2010-15/04/2010*, http ://www.acm.org/, avril 2010. ACM.

[CCD+05] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, et al. Grid'5000 : a large scale and highly reconfigurable grid experimental testbed. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, page 99–106, Seattle, Washington, USA, 2005.

[CCG+05] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard. A batch scheduler with high level components. In *Cluster Computing and the Grid, 2005. CCGrid 2005.*, 2005.

[CD06] E. Caron and F. Desprez. DIET : a scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3) :335, 2006.

[Che13] Ghislain Landry Tsafack Chetsa. *System Profiling and Green Capabilities for Large Scale and Distributed Infrastructures.* PhD thesis, Ecole Normale Supérieure de Lyon, 2013.

[CMR96] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundam. Inf.*, 26(3-4) :241–265, June 1996.

[CMR+97] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. *Handbook of Graph Grammars and Computing by Graph Transformations*, chapter Algebraic Approaches to Graph Transformation. Part I : Basic Concepts and Double Pushout Approach, pages 163–245. Volume 1 : Foundations of Rozenberg [Roz97], 1997.

[CMRB14] R. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Transactions on Cloud Computing*, (99), 2014.

[CRIrg] CRIU. Checkpoint/Restore In Userspace. `http://www.criu.org/`.

[ctr] Projet anr ctrl green. `http://www.ctrlgreen.org/`.

[Da 13] Da Costa, G. Heterogeneity : The Key to Achieve Power-Proportional Computing. *IEEE Int. Symp. on Cluster Computing and the Grid*, 2013.

[DCP15] Georges Da Costa and Jean-Marc Pierson. DVFS governor for HPC : Higher, Faster, Greener. In *Parallel, Distributed, and Network-Based Processing (PDP), Turku, Finland, 04/03/2015-06/03/2015*, pages 533–540, http ://www.ieee.org/, mars 2015. IEEE.

[DCZJO13] G. Da Costa, T. Zilio, M. Jarus, and A. Oleksiak. Energy-and heat-aware HPC benchmarks. In *Proceedings of the International Conference on Cloud and Green Computing (CGC)*, pages 435–442, 2013.

[DGLM13] Mohammed el Mehdi Diouri, Olivier Gluck, Laurent Lefevre, and Jean-Christophe Mignot. Your cluster is not power homogeneous : Take care when designing green schedulers ! 2013.

[DNP13] J.J. Durillo, V. Nae, and R. Prodan. Multi-objective workflow scheduling : An analysis of the energy efficiency and makespan tradeoff. In *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 203–210, 2013.

[DSH+13] Z. Du, H. Sun, Y. He, Y. He, D.A. Bader, and H. Zhang. Energy-efficient scheduling for best-effort interactive services to achieve high response quality. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pages 637–748, 2013.

[DTCG+13] Mehdi. Diouri, Ghislain Landry Tsafack Chetsa, Olivier Gluck, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf, and Georges Da Costa. Energy efficiency in HPC : with or without knowledge on applications and services. *International Journal of High Performance Computing Applications*, doi :10.1177/1094342013497990 :232–243, août 2013.

[DtH01] M. Dumas and A. ter Hofstede. UML activity diagrams as a workflow specification language. *UML 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, page 76–90, 2001.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental theory for typed attributed graphs and graph transformation based on adhesive hlr categories. *Fundamenta Informaticae*, 74(1) :pp. 31–61, October 2006.

[EGM+14] Cédric Eichler, Ghada Gharbi, Thierry Monteil, Nawal Guermouche, and Patricia Stolf. Graph-based formalism for Machine-to-Machine self-managed communications (regular paper). In *IEEE International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE), Hammamet (Tunisie), 17/06/2013-22/06/2013*, pages 74–79, http ://www.ieee.org/, juin 2014. IEEE.

[EHK+97] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. *Handbook of Graph Grammars and Computing by Graph Transformations*, chapter Algebraic Approaches to Graph Transformation. Part II : Single Pushout Approach and Comparison with Double Pushout Approach, pages 247–312. Volume 1 : Foundations of Rozenberg [Roz97], 1997.

[Eic15] Cédric Eichler. *Modélisation formelle de systèmes dynamiques autonomes : graphe, réécriture et grammaire*. PhD thesis, Université Toulouse 3 Paul Sabatier, 2015.

[EJF14] Khosrow Ebrahimi, Gerard F. Jones, and Amy S. Fleischer. A review of data center cooling technology, operating conditions and the corresponding low-grade waste heat recovery opportunities. *Renewable and Sustainable Energy Reviews*, 31 :622–638, march 2014.

[EMS+14] Cédric Eichler, Thierry Monteil, Patricia Stolf, Alfredo Grieco, and Khalil Drira. Enhanced Graph Rewriting Systems for Complex Software Domain. *Software and Systems Modeling*, 10.1007/s10270-014-0433-1 :1–21, septembre 2014.

[ERDPB+13] Ahmed El Rheddane, Noel De Palma, Fabienne Boyer, Frédéric Dumont, Jean-Marc Menaud, and Alain Tchana. Dynamic Scalability of a Consolidation Service. In *International Conference on Cloud Computing, Indus Track*, pages 01–09, United States, June 2013.

[FCDCO+15] Leandro Fontoura Cupertino, Georges Da Costa, Ariel Oleksiak, Wojciech PiaTek, Jean-Marc Pierson, Jaume Salom, Laura Siso, Patricia Stolf, Hongyang Sun, and Thomas Zilio. Energy-Efficient, Thermal-Aware Modeling and Simulation of Datacenters : The CoolEmAll Approach and Evaluation Results. *Ad Hoc Networks Journal*, 2015.

[Fel12] Eugen Feller. *Autonomic and energy-efficient management of large scale virtualized Data Centers.* PhD thesis, Université Rennes 1, 2012.

[FKLB08] V.W. Freeh, N. Kappiah, D.K. Lowenthal, and T.K. Bletsch. Just-in-time dynamic voltage scaling : Exploiting inter-node slack to save energy in MPI programs. *Journal of Parallel and Distributed Computing*, 68(9) :1175–118, 2008.

[FL05] Vincent W. Freeh and David K. Lowenthal. Using multiple energy gears in mpi programs on a power-scalable cluster. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '05, pages 164–173, New York, NY, USA, 2005. ACM.

[Fon15] Leandro Fontoura. *Modeling the power consumption of computing systems and applications through Machine Learning techniques.* PhD thesis, Université Toulouse 3 Paul Sabatier, 2015.

[FPBF12] H.M. Fard, R. Prodan, J. Barrionuevo, and T. Fahringer. A multi-objective approach for workflow scheduling in heterogeneous environments. In *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 300–309, 2012.

[fre] Free cooling. `www.ecoinfo.cnrs.fr/article140.html`.

[gam] Games research project. `http://www.green-datacenters.eu`.

[GCCD+15] Alfonso Gazo Cervero, Michele Chincoli, Laars Dittmann, Andreas Fischer, Alberto E. Garcia, Laurent Lefèvre, Gergö Lovász, Hermann De Meer, Paolo Monti, Anne-Cécile Orgerie, Louis-François Pau, Chris Phillips, Patricia Stolf, Luca Valcarenghi, Thierry Monteil, Remi Sharrock, Tuan Trinh, and Sergio Ricciardi. Green Wired Networks. In Jean-Marc Pierson, editor, *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*, chapter 3, pages 41–80. John Wiley and Sons, http ://www.wiley.com, mai 2015.

[GCS] Green cloud scheduler. `http://coned.utcluj.ro/GreenCloudScheduler/Documentation.html#`.

[GDCR15] Davide Gareglio, Georges Da Costa, and Sergio Ricciardi. Hardware leverages for energy reduction in large-scale distributed systems. In Jean-Marc Pierson, editor, *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*, chapter 2, pages 17–39. John Wiley and Sons, http ://www.wiley.com, mai 2015. ISBN : 978-1-118-86463-0.

[GGQ+13] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79(8) :1230–1242, 2013.

[GHZ13] C. Ghribi, M. Hadji, and D. Zeghlache. Energy efficient vm scheduling for cloud data centers : Exact allocation and migration algorithms. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 671–678, May 2013.

[gou] Dvfs : Gouverneurs linux. `www.kernel.org/doc/Documentation/cpu-freq/governors.txt`.

[Grea] Green500. `www.green500.org`.

[Greb] Harmonizing global metrics for data center energy efficiency, global task force reaches agreement regarding data center productivity. `www.thegreengrid.org/~/media/Regulatory/\HarmonizingGlobalMetricsforDataCenterEnergyEfficiency.pdf?lang=en`.

[GS12] R. Garg and A.K. Singh. Reference point based multi-objective optimization to workflow grid scheduling. *International Journal of Applied Evolutionary Computation*, 3(1) :80–99, 2012.

[Gue07] Mohammed Karim Guennoun. *Architectures Dynamiques dans le cadre des applications à base de composants et orientées services.* PhD thesis, université Toulouse III-Paul Sabatier, 2007.

[Gue14] Tom Guerout. *Ordonnancement sous contraintes de qualité de serivce dans les clouds. Networking and Internet Architecture.* PhD thesis, INSA Toulouse, 2014.

[GUI94] J. P LE GUIGNER. RENATER : réseau national de la technologie, de l'enseignement et de la recherche. *Bulletin des bibliothèques de France*, 39(1) :39–44, 1994.

[GYAB11] S.K. Garg, C. Yeo, A. Anandasivam, and R. Buyya. Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6) :732–749, 2011.

[Hem] Hemera. `www.grid5000.fr/mediawiki/index.php/Hemera`.

[HIM99] Dan Hirsch, Paola Inverardi, and Ugo Montanari. Modeling software architectures and styles with graph grammars and constraint solving. In Patrick Donohoe, editor, *Software Architecture*, volume 12 of *The International Federation for Information Processing*, pages 127–143. Springer US, 1999.

[HK03] Chung-Hsing Hsu and Ulrich Kremer. The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, PLDI '03, pages 38–48, New York, NY, USA, 2003. ACM.

[HKJHKD05] Mohamed Hadj Kacem, Mohamed Jmaiel, Ahmed Hadj Kacem, and Khalil Drira. Evaluation and comparison of ADL based approaches for the description of dynamic of software architectures. In *7th International Confe-*

rence on Enterprise Information Systems (ICEIS), pages 189–195, Miami, USA, 2005.

[HLM+09] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia L. Lawall. Entropy : a Consolidation Manager for Clusters. In *VEE '09 : Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 41–50, New York, NY, United States, March 2009. ACM.

[HLS+14] Jin Hai, Deng Li, Wu Song, Shi Xuanhua, Chen Hanhua, and Pan Xiaodong. MECOM : Live migration of virtual machines by adaptively compressing memory pages. *Future Generation Computer Systems*, 38, 2014.

[HM00] Dan Hirsch and Ugo Montanari. Consistent transformations for software architecture styles of distributed systems. *Electronic Notes in Theoretical Computer Science*, 28(0) :4–25, 2000.

[Hor01] Paul Horn. Autonomic computing : Ibm's perspective on the state of information technology. Technical report, IBM white papers, 2001.

[how] How clean is your cloud ? `http://www.greenpeace.org`.

[ibm09] IBM - mathematical programs - IBM ILOG CPLEX optimizer - software. http ://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, 2009.

[ICM06] Canturk Isci, Gilberto Contreras, and Margaret Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 359–370, Washington, DC, USA, 2006. IEEE Computer Society.

[ITU09] ITU. Background paper for the itu symposium on icts and climate change. In *ITU Symposia on ICTs and Climate Change*, 2009.

[JJFHHCLD14] Jheng Jhu-Jyun, Tseng Fan-Hsun, Chao Han-Chieh, and Chou Li-Der. A novel VM workload prediction using Grey Forecasting model in cloud data center. In *2014 International Conference on Information Networking (ICOIN)*, 2014.

[jop05] JOpt, a simplified java wrapper for linear and mixed integer programming. http ://www.eecs.harvard.edu/econcs/jopt/, 2005.

[JR] La troisième révolution industrielle. `http://www.latroisiemerevolutionindustrielleennordpasdecalais.fr/jeremy-rifkin/`.

[JSPJJ12] Yang Jyun-Shiung, Liu Pangfeng, and Wu Jan-Jan. Workload characteristics-aware virtual machine consolidation algorithms. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 42–49, Dec 2012.

[KC03] J.O. Kephart and D.M Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, 2003.

[KFL05] N. Kappiah, Vincent W. Freeh, and D.K. Lowenthal. Just in time dynamic voltage scaling : Exploiting inter-node slack to save energy in mpi programs. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 33–33, 2005.

[KGWB08] W. Kim, M.S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA*, 2008.

[KIS10] Hideaki Kimura, Takayuki Imada, and Mitsuhisa Sato. Runtime energy adaptation with low-impact instrumented code in a power-scalable cluster system. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 378–387, Washington, DC, USA, 2010. IEEE Computer Society.

[KMT13] Y. Kessaci, N. Melab, and E-G. Talbi. A Pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation. *Cluster Computing*, 16 :451–468, 2013.

[KMT14] Y. Kessaci, N. Melab, and E-G. Talbi. A multi-start local search heuristic for an energy efficient VMs assignment on top of the OpenNebula cloud manager. *Future Generation Computer Systems*, 36 :237–256, 2014.

[Koo11] J. Koomey. Growth in data center electricity use 2005 to 2010. 2011.

[KOP+13] Krzysztof Kurowski, Ariel Oleksiak, Wojciech Piatek, Tomasz Piontek, Andrzej W. Przybyszewski, and Jan Weglarz. Dcworms - a tool for simulation of energy efficiency in distributed computing infrastructures. *Simulation Modelling Practice and Theory*, 39 :135–151, 2013.

[Lö93] Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109(1–2) :181 – 224, 1993.

[LFL06] Min Yeol Lim, Vincent W. Freeh, and David K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[LHKJD04] I. Loulou, A. Hadj Kacem, M. Jmaiel, and K. Drira. Towards a unified graph-based framework for dynamic component-based architectures description in Z. In *Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS)*, pages 227–234, July 2004.

[LKA+95] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann. Specification and analysis of system architecture using rapide. *IEEE Trans. Software Eng.*, 21(4) :336–355, 1995.

[LKK99] W. Leinberger, G. Karypis, and V. Kumar. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *Parallel Processing, 1999. Proceedings. 1999 International Conference on*, pages 404–412, 1999.

[LM96] Daniel Le Métayer. Software Architecture Styles As Graph Grammars. *SIGSOFT Softw. Eng. Notes*, 21(6) :15–23, October 1996.

[LO10] Laurent Lefèvre and Anne-Cécile Orgerie. Designing and Evaluating an Energy Efficient Cloud. *Journal of Supercomputing*, 51(3) :352–373, March 2010.

[LPP04] S. Lacour, C. Pérez, and T. Priol. A network topology description model for grid application deployment. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, page 61–68, 2004.

[LS06] X. Lin and N. B. Shroff. Utility maximization for communication networks with multipath routing. *IEEE Transactions on Automatic Control*, 51(5), 2006.

[LVP+15] Laurent Lefèvre, Sebastien Varrette, Frédéric Pinel, Pascal Bouvry, Ghislain Landry Tsafack Chetsa, Georges Da Costa, Patricia Stolf, Jean-Marc Pierson, and Emmanuel Jeannot. Energy Efficiency and High Performance Computing. In Jean-Marc Pierson, editor, *Large-Scale Distributed Systems and Energy Efficiency : A holistic view*, chapter 7, pages 197–224. John Wiley and Sons, http ://www.wiley.com, mai 2015. ISBN : 978-1-118-86463-0.

[LZ09] Y.C. Lee and A. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *CCGrid*, 2009.

[LZ10] Yongpeng Liu and Hong Zhu. A survey of the research on power management techniques for high-performance systems. *Software Practice and Experience*, 40(11) :943–964, Oct 2010.

[LZ11] Y. Lee and A.Y. Zomaya. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel and Distributed Systems*, 22(8) :1374–1381, 2011.

[MC06] J. Moore and P. Chase, J.S.and Ranganathan. Weatherman : Automated, online and predictive thermal mapping and management for data centers. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 155–164, 2006.

[MCRS05] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling “cool" : temperature-aware workload placement in data centers. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC)*, 2005.

[MK96] Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. In *4th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 3–14, 1996.

[MOC+14] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athnasios V. Vasilakos. Cloud Computing : Survey on Energy Efficiency. *ACM Computing Surveys*, 47(2) :1–36, décembre 2014.

[ODdAL14] Anne-Cécile Orgerie, Marcos Dias de Assunção, and Laurent Lefèvre. Acm computing surveys. *A Survey on Techniques for Improving the Energy Efficiency of Large Scale Distributed Systems*, 46(4), April 2014.

[ONE14] Open nebula : Match making scheduler. `http://archives.opennebula.org/documentation:rel4.4:schg`, June 2014.

[Oqu04] Flavio Oquendo. π-ARL : an architecture refinement language for formally modelling the stepwise refinement of software architectures. *SIGSOFT Softw. Eng. Notes*, 29(5) :1–20, September 2004.

[Oqu08] F. Oquendo. Dynamic software architectures : Formally modelling structure and behaviour with Pi-ADL. In *"The Third International Conference on Software Engineering Advances"*, pages 352–359, Oct 2008.

[OTS+11] Aurelien Ortiz, François Thiébolt, Patricia Stolf, Georges Da Costa, and Amal Sayah. Virtual Machine Migration : A Comparative Study of Storage Viewpoints (regular paper). In *International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Ajaccio, Corsica, France, 12/04/2011-15/04/2011*, page (electronic medium), http ://www.civil-comp.com/conf/progp2011.htm, avril 2011. Civil-Comp Proceedings.

[Pas04] Patricia Pascal. *Gestion de ressources pour des services déportés sur des grappes d'ordinateurs avec qualité de service garantie.* PhD thesis, INSA Toulouse, 2004.

[Pas14] Passmark results. `http://www.cpubenchmark.net`, 2014.

[PCL+11] V. Petrucci, E. V. Carrera, O. Loques, J. C. B. Leite, and D. Mosse. Optimized management of power and performance for virtualized heterogeneous server clusters. In *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 23–32, 2011.

[PF11] Terence Parr and Kathleen Fisher. Ll(*) : the foundation of the antlr parser generator. *SIGPLAN Not.*, 47(6) :pp.425–436, June 2011.

[PMWV09] S. Pelley, D. Meisner, T. Wenisch, and J. Vangilder. Understanding and abstracting total data center power. pages 2706–2710, 2009.

[Poe06] I. Poernomo. The meta-object facility typed. In *Proceedings of the 2006 ACM symposium on Applied computing*, page 1849, 2006.

[PP09] E. Pakbaznia and M. Pedram. Minimizing data center cooling and server power costs. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 145–150, 2009.

[PRM+12] Carlos Parra, Daniel Romero, Sébastien Mosser, Romain Rouvoy, Laurence Duchien, and Lionel Seinturier. Using constraint-based optimization and variability to support continuous self-adaptation. In *27th Annual ACM Symposium on Applied Computing*, pages 486–491, 2012.

[RDC+10] Ismael Bouassida Rodriguez, Khalil Drira, Christophe Chassot, Karim Guennoun, and Mohamed Jmaiel. A rule driven approach for architectural self adaptation in collaborative activities using graph grammars. *Int. J. Autonomic Comput.*, 1(3) :226–245, May 2010.

[REC14] The RECS testbed. `http://recs.irit.fr/doku.php?id=recs:features`, June 2014.

[Ren] Arend Rensing. *The GROOVE Simulator : A Tool for State Space Generation*, pages 479–485.

[RLdS+09] Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio : making dvs practical for complex hpc applications. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM.

[RLW13] A. Rahim, B. Lukman, and J. Whittle. A survey of approaches for verifying model transformations. *Software & Systems Modeling*, pages 1–26, 2013.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1 : Foundations. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.

[RPE11] Ryckbosch, Polfliet, and Eeckhout. Trends in Server Energy Proportionality. *Computer*, 2011.

[RTZL10] N.B. Rizvandi, J. Taheri, A.Y. Zomaya, and Y. Lee. Linear combinations of dvfs-enabled processor frequencies to modify the energy-aware scheduling algorithms. In *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 388–397, 2010.

[SA12] H.F. Sheikh and I. Ahmad. Simultaneous optimization of performance, energy and temperature for dag scheduling in multi-core processors. In *Proceedings of the International Green Computing Conference (IGCC)*, pages 1–6, 2012.

[Saw04] R. Sawyer. Calculating total power requirements for data centers, 2004.

[SAWR12] H. F. Sheikha, I. Ahmada, Z. Wang, and S. Rankab. An overview and classification of thermal-aware scheduling techniques for multi-core processing systems. *Sustainable Computing : Informatics and Systems*, 2 :151—-169, 2012.

[SCH09] H. Sun, Y. Cao, and W.-J. Hsu. Non-clairvoyant speed scaling for batched parallel jobs on multiprocessors. In *Proceedings of the ACM Conference on Computing Frontiers*, pages 99–108, 2009.

[Sco93] R. S. Scowen. Extended BNF-a generic base standard. In *Software Engineering Standards Symposium*, volume 3, page 6–2, 1993.

[Sha10] Remi Sharrock. *Gestion autonomique de performance, d'énergie et de qualité de service. Application aux réseaux filaires, réseaux de capteurs et grilles de calcul*. PhD thesis, Institut National Polytechnique de Toulouse - INPT, December 2010.

[SKD+05] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, W. Wang, and J. G. Powers. A description of the advanced research WRF version 2. *NCAR Tech Note*, NCAR/TN-468+STR, 2005.

[SKK11] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors : A framework for continuously adaptive dvfs. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8, 2011.

[SKM+09] Remi Sharrock, Fadi Khalil, Thierry Monteil, Hervé Aubert, Fabio Coccetti, Patricia Stolf, Laurent Broto, and Robert Plana. Deployment and management of large planar reflectarray antennas simulation on grid (regular paper). In *Challenges of Large Applications in Distributed Environments, Workshop in conjunction with HPDC 2009 (CLADE), Munich, 09/06/2009-10/06/2009*, pages 17–26, http ://www.acm.org/, 2009. ACM.

[SMS+11] Rémi Sharrock, Thierry Monteil, Patricia Stolf, Daniel Hagimont, and Laurent Broto. Non-Intrusive Autonomic Approach with Self-Management Policies Applied to Legacy Infrastructures for Performance Improvements. *Int. J. Adapt. Resilient Auton. Syst.*, 2(1) :58–76, January 2011.

[SMSB13] Remi Sharrock, Thierry Monteil, Patricia Stolf, and Olivier Brun. Autonomic computing to manage green Core networks with Quality of Service (regular paper). In *Energy Efficiency in Large Scale Distributed Systems conference (EE-LSDS), Vienna, 22/04/2013-24/04/2013*, pages 248–263, http ://www.springerlink.com, 2013. Springer.

[Sno] Snooze : Energy-efficient cloud manager. `www.snooze.inria.fr`.

[SOP] Video du démontrateur du projet anr sop. `https://youtu.be/PZwgUu-XLqY`.

[SPE12] Standard performance evaluation corporation, spec power benchmark results. `http://www.spec.org/power_ssj2008/results/res2012q1/`, march 2012.

[SPE14] Standard performance evaluation corporation, specpower_ssj2008 results. `http://www.spec.org/power_ssj2008/results/`, October 2014.

[SSH+03] K. Skadron, M.R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *ISCA*, 2003.

[SSJ+13] L. Siso, J. Salom, M. Jarus, A. Oleksiak, and T. Zilio. Energy and heat-aware metrics for data centers : Metrics analysis in the framework of coolemall project. In *Third International Conference on Cloud and Green Computing (CGC)*, pages 428–434, 2013.

[SSM10] Remi Sharrock, Patricia Stolf, and Thierry Monteil. Extending TUNe for autonomous management of QoS at application and network levels (short paper). In *International Conference on Networks (ICN), Menuires, France, 11/04/2010-16/04/2010*, page (electronic medium), http ://www.ieee.org/, 2010. IEEE.

[SSMG11] Remi Sharrock, Patricia Stolf, Thierry Monteil, and Tom Guerout. Internal self-protecting for consistency and stability in an autonomic manager (regular paper). In *IEEE International Symposium on Network Computing and Applications (IEEE NCA), Toulouse, 21/10/2011-23/10/2011*, ISBN : 978-0-7695-4550-9, pages 44–49, http ://www.computer.org, 2011. IEEE Computer Society.

[SSPDC14a] Hongyang Sun, Patricia Stolf, Jean-Marc Pierson, and Georges Da Costa. Energy-efficient and thermal-aware resource management for heterogeneous datacenters. *Sustainable Computing : Informatics and Systems*, 10.1016/j.suscom.2014.08.005 :1–15, août 2014.

[SSPDC14b] Hongyang Sun, Patricia Stolf, Jean-Marc Pierson, and 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, 26/05/14-29/05/14*, page (electronic medium), http ://www.computer.org, mai 2014. IEEE Computer Society.

[SX03] Petri Selonen and Jianli Xu. Validating UML models against architectural profiles. *SIGSOFT Software Engineering Notes*, 28 :pp.58–67, September 2003.

[Tae] Grabriele Taentzer. *AGG : A Graph Transformation Environment for Modeling and Validation of Software*, pages 446–453.

[TCLP+12a] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf, and 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*, pages 132–138, http ://www.computer.org/portal/web/cscps, octobre 2012. IEEE Computer Society - Conference Publishing Services.

[TCLP+12b] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf, and 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*, pages 141–152, http ://www.springerlink.com, mai 2012. Springer.

[TCLP+13] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf, and 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*, pages 118–125, http ://www.computer.org, août 2013. IEEE Computer Society.

[TCLP+14a] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf, and 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*, http ://www.computer.org/portal/web/cscps/home, 2014. CPS (Conference Publishing Services).

[TCLP+14b] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf, and Georges Da Costa. Exploiting performance counters to predict and improve energy performance of HPC systems. *Future Generation Computer Systems*, 10.1016/j.future.2013.07.010 :287–298, juillet 2014.

[TCLS13] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, and Patricia Stolf. A Three Step Blind Approach for Improving HPC Systems' Energy Performance (regular paper). In *Energy Efficiency in Large Scale Distributed Systems conference (EE-LSDS), Vienna, 22/04/2013-24/04/2013*, pages 168–181, http ://www.springerlink.com, 2013. Springer.

[TCLS14] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, and Patricia Stolf. A Three Step Blind Approach for Improving HPC Systems' Energy Performance. *Concurrency and Computation :Practice and Experience*, 10.1002/cpe.3312 :1–18, juillet 2014.

[TGV08] Q. Tang, S. K. S. Gupta, and G. Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers : A cyber-physical approach. *IEEE Transactions on Parallel and Distributed Systems*, 19(11) :1458–1472, 2008.

[TMGC06] Q. Tang, T. Mukherjee, S. Gupta, and P. Cayton. Sensor-based fast thermal evaluation model For energy efficient high-performance datacenters. In *Proceedings of the International Conference on Intelligent Sensing and Information Processing*, pages 203–208, 2006.

[Top] Top500. `www.top500.org`.

[Tou10] Hélène Toussaint. *Algorithmique rapide pour les problèmes de tournées et d'ordonnancement*. PhD thesis, Université Blaise Pascal - Clermont-Ferrand II, 2010.

[TTS+12] S. Takahashi, A. Takefusa, M. Shigeno, H. Nakada, T. Kudoh, and A. Yoshise. Virtual machine packing algorithms for lower power consumption. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 161–168, Dec 2012.

[Upt] Uptime institute, 2014 data center industry survey. `http://journal.uptimeinstitute.com/2014-data-center-industry-survey/`.

[VAG10] Varsamopoulos, Abbasi, and Gupta. Trends and Effects of Energy Proportionality on Server Provisioning in Data Centers. In *Int. Conf. on High Performance Computing*, 2010.

[VDCL+14] Violaine Villebonnet, Georges Da Costa, Laurent Lefèvre, Jean-Marc Pierson, and 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*, pages 703–710, http ://www.ieee.org/, décembre 2014. IEEE.

[VDCL+15] V. Villebonnet, G. Da Costa, L. Lefevre, JM. Pierson, and P. Stolf. Big, medium, little : Reaching energy proportionality with heterogeneous computing scheduler. *Parallel Processing Letters (à paraître)*, 2015.

[VLK+13] G. L. Valentini, W. Lassonde, S. U. Khan, N. Min-Allah, S. A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A. Y. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, and P. Bouvry. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16 :3–15, 2013.

[VTG+13] E. Volk, A. Tenschert, M. Gienger, A. Oleksiak, L. Siso, and J. Salom. Improving energy efficiency in data centers and federated cloud environments : Comparison of coolemall and eco2clouds approaches and metrics.

In *Third International Conference on Cloud and Green Computing (CGC)*, pages 443–450, 2013.

[WF02] M. Wermelinger and J. L. Fiadeiro. A graph transformation approach to software architecture reconfiguration. *Science of Computer Programming*, 44(2) :133–155, 2002. Special Issue on Applications of Graph Transformations (GRATRA 2000).

[Wil01] David Wile. Using dynamic ACME. In *Working Conference on Complex and Dynamic Systems Architecture*, 2001.

[WKD12] L. Wang, S.U. Khan, and J. Dayal. Thermal aware workload placement with task-temperature profiles in a data center. *The Journal of Supercomputing*, 61(3), 2012.

[WvLD+09] L. Wang, G. von Laszewski, J. Dayal, X. He, A.J. Younge, and T.R. Furlani. Towards thermal aware workload scheduling in a data center. In *Proceedings of the International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)*, pages 116–122, 2009.

[XF10] J. Xu and J. A. B. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Proceedings of the IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing (GREENCOM-CPSCOM)*, pages 179–188, 2010.

[XSC13] Zhen Xiao, Weijia Song, and Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6) :1107–1117, June 2013.

[YKB07] J. Yu, M. Kirley, and R. Buyya. Multi-objective planning for workflow execution on grids. In *Proceedings of the IEEE/ACM International Conference on Grid Computing*, pages 10–17, 2007.

[ZC07] S. Zhang and K.S. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *ICCAD*, 2007.

[ZCL+13] F. Zhang, J. Cao, K. Li, S.U. Khan, and K. Hwang. Multi-objective scheduling of many tasks in cloud platforms. *Future Generation Computer Systems*, 2013.