Efficacité énergétique dans le calcul très haute performance : application à la tolérance aux pannes et à la diffusion de données

Mohammed El Mehdi Diouri

NNT: 2013ENSL0836

THÈSE

en vue d'obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE LYON
délivré par
l'ÉCOLE NORMALE SUPÉRIEURE DE LYON
Spécialité : Informatique

LABORATOIRE DE L'INFORMATIQUE DU PARALLÉLISME
École Doctorale Informatique et Mathématiques (ED 512)

présentée et soutenue publiquement le 27 Septembre 2013 par

Monsieur **Mohammed El Mehdi DIOURI**

Titre :

Éfficacité énergétique dans le calcul très haute performance : application à la tolérance aux pannes et à la diffusion de données

---

Directrice de thèse :
Madame ISABELLE GUÉRIN-LASSOUS

Après avis de :
Monsieur CHRISTOPHE CÉRIN
Monsieur PIERRE SENS

Devant le Jury composé de :

Monsieur	CHRISTOPHE CÉRIN	Rapporteur
Monsieur	OLIVIER GLÜCK	Co-encadrant
Madame	ISABELLE GUÉRIN-LASSOUS	Directrice
Monsieur	LAURENT LEFÈVRE	Co-encadrant
Monsieur	JEAN-FRANÇOIS MÉHAUT	Président
Monsieur	PIERRE SENS	Rapporteur

Remerciements

Tout d'abord, je tiens à remercier très sincèrement M. Olivier Glück (Maître de Conférences à l'Université Claude Bernard Lyon 1) et M. Laurent Lefèvre (Chargé de Recherches à l'INRIA) pour leur encadrement, leur disponibilité et leurs nombreux conseils très pédagogiques qui m'ont été d'une grande utilité. Je remercie également Mme Isabelle Guérin-Lassous (Professeur à l'Université Claude Bernard Lyon 1) pour avoir accepté de diriger ma thèse, mais aussi pour son écoute et sa disponibilité tout au long de ma thèse.

J'adresse également mes remerciements à M. Christophe Cérin (Professeur à l'Université Paris XIII) et M. Pierre Sens (Professeur à l'Université Paris VI) pour avoir accepté de rapporter sur ma thèse et pour l'intérêt qu'ils ont manifesté pour mon travail. Mes remerciements vont également à M. Jean-François Méhaut (Professeur à l'Université Joseph Fourier Grenoble 1) pour avoir accepté d'être membre de mon jury de thèse.

Merci à M. Franck Cappello (Directeur de Recherches à l'INRIA et Chercheur associé à l'Université de l'Illinois à Urbana-Champaign) pour ses conseils très pertinents ainsi que pour le travail de recherche effectué avec lui. Je remercie aussi M. Jean-Christophe Mignot (Ingénieur de Recherches au CNRS) pour toutes les discussions intéressantes que j'ai eues avec lui durant ma thèse.

Je tiens à remercier M. Eddy Caron, M. Olivier Glück, Mme Anne Benoît, Mme Isabelle Guérin-Lassous et M. Nicolas Lumineau pour m'avoir confié des tâches d'enseignement dans le cadre de mon monitorat.

Merci à tous les anciens membres de l'équipe RESO et à tous les membres de l'équipe AVALON, et en particulier aux responsables respectifs de ses deux équipes de recherche, M. Paulo Gonçalves (Chargé de Recherches à l'INRIA) et M. Christian Pérez (Directeur de Recherches à l'INRIA), et aux assistantes respectives de ses deux équipes de recherche, Mme Laetitia Lecot et Mme Evelyne Blesle, pour leur aide relative à l'organisation de mes missions. Merci également à tous les anciens doctorants et tous les doctorants du laboratoire LIP avec qui j'ai passé des moments conviviaux pendant mes trois années de thèse.

Enfin, je tiens à exprimer ma gratitude et ma reconnaissance à mes parents pour leur soutien et leurs encouragements et cela depuis mon plus jeune âge. Je tiens aussi à remercier ma femme, mon frère ainsi que toute ma famille pour les moments agréables que j'ai passés avec eux et pour leurs encouragements durant la période de ma thèse.

À mes parents,

À ma femme,

À mon frère,

À toute ma famille.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Un supercalculateur est une infrastructure construite à partir d'une interconnexion d'ordinateurs capables d'effectuer des tâches en parallèle dans l'optique d'atteindre des performances globales d'exécution très élevées. Depuis le début des années 90, ces infrastructures de calcul très haute performance ont connu une croissance rapide. En effet, on peut constater grâce au TOP500 [1] [Meuer, 2000], qu'environ tous les 11 ans, les performances des superordinateurs connaissent une croissance d'un facteur 1000. La métrique qui permet de classer les supercalculateurs les plus performants est le FLOPS, qui est le nombre d'opérations à virgule flottante par seconde lors de l'exécution d'un banc d'essai particulier : le LINPACK [2] [Dongarra, 1987]. La figure 1.1 illustre cette croissance des performances des supercalculateurs.

Un gigaFLOPS a été atteint en 1985 par le système Cray 2 [3], un téraFLOPS a été atteint en 1997 par le système ASCI Red [4], un pétaFLOPS a été atteint en 2008 par Roadrunner [5]. Selon la liste TOP500 publiée en juin 2013, le superordinateur le plus puissant est Tianhe-2 (MilkyWay-2) [6] une machine comportant plus de 3 millions de coeurs et capable d'atteindre plus de 33.86 pétaFLOPS. Cette tendance linéaire de la croissance de la performance n'a pas cessé et est loin de connaître un quelconque ralentissement vu qu'il est prévu de franchir le seuil de l'exaFLOPS d'ici 2018 (voir figure 1.1).

Depuis 2003, la fréquence des processeurs n'augmente plus car la limite de puissance électrique dissipée par une puce a été atteinte. Afin de maîtriser au mieux cette puissance électrique dissipée (directement liée à la fréquence d'horloge), les constructeurs favorisent le développement des processeurs multi-coeurs. Pour atteindre des performances de calcul aussi élevées, les plates-formes de calcul atteignant un exa-

1. TOP500 [www.top500.org] : la liste qui classe les supercalculateurs les plus puissants au monde
2. LINPACK : http://www.top500.org/project/linpack/
3. Cray 2 : http://archive.computerhistory.org/resources/text/Cray/Cray.Cray2.1985.102646185.pdf
4. ASCI Red : http://www.sandia.gov/ASCI/Red/index.html
5. Roadrunner : http://www.lanl.gov/roadrunner/
6. Tianhe-2 : http://www.nudt.edu.cn/Articleshow_eng.asp?id=6242

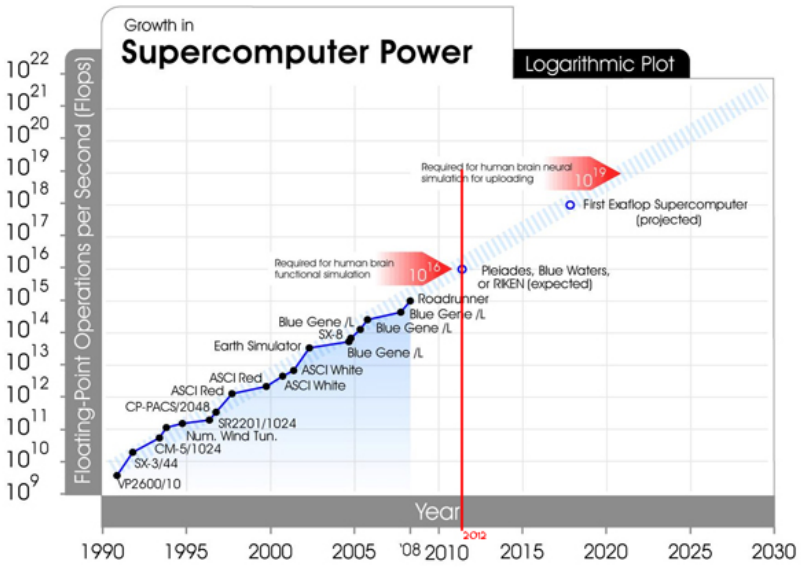

FIGURE 1.1 – Évolution et prédiction de la performance maximale atteinte par les supercalculateurs (extrait de www.top500.org)

FLOPS devraient comporter typiquement entre un demi-million et plusieurs millions de coeurs de calcul exécutant jusqu'à un milliard de processus [Cappello et al., 2009].

Par ailleurs, les supercalculateurs sont devenus de gros consommateurs d'électricité. Au delà des problèmes écologiques, les coûts financiers liés à ces infrastructures sont un véritable frein à leur déploiement. Ces coûts très élevés incluant l'installation des infrastructures et les coûts d'entretien deviennent des facteurs prédominants dans le coût total de possession d'un superordinateur. Les 5 supercalculateurs les plus performants d'après le TOP500 consomment déjà plusieurs mégawatts. Un mégawatt représente la puissance électrique moyenne dissipée par un village de 1200 habitants en France (Consommation 2012 de 489 Twh pour la France). Bien qu'ils aient des performances équivalentes (5 pétaFLOPS), les supercalculateurs Stampede et Juqueen ont des consommations électriques très différentes : Juqueen consomme 2.3 MW alors que Stampede consomme 4.5 MW, ce qui représente quasiment le double !

Pour évaluer l'efficacité énergétique des supercalculateurs, un autre classement existe depuis avril 2005 : la liste GREEN 500 [7] [Feng et al., 2010, Sharma et al., 2006]. Cette liste introduit la métrique du FLOPS par watt (FLOPS/W) qui prend en compte la consommation électrique relativement à la puissance de calcul de la

7. GREEN 500 [www.green500.org] : la liste qui classe les supercalculateurs les plus efficaces en énergie

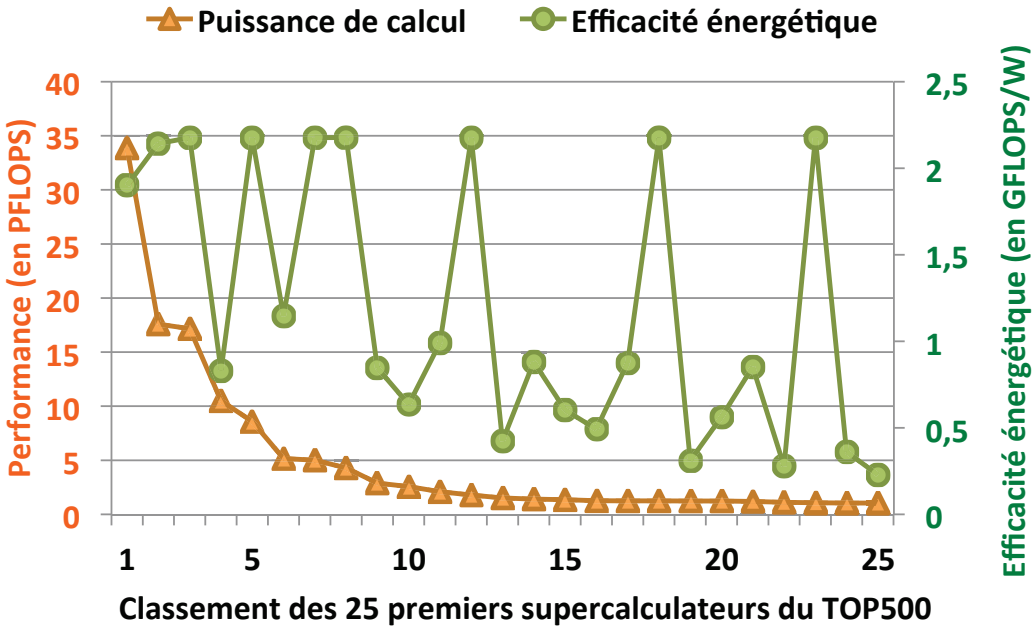

FIGURE 1.2 – Efficacités énergétiques des 25 premiers supercalculateurs selon le TOP500 (données extraites de www.green500.org et de www.top500.org)

machine. En s'appuyant sur les données extraites de la liste du GREEN 500 datant de juin 2013, la figure 1.2 présente les efficacités énergétiques des 25 premiers supercalculateurs du TOP500. Cette figure permet de noter que l'efficacité énergétique n'est pas forcément liée à la performance. D'après ce même classement, les deux supercalculateurs les plus efficaces énergétiquement sont Eurora et Aurora Tigon dans la mesure où ils atteignent respectivement 3.209 et 3.180 gigaFLOPS par watt. Cependant, leurs puissances maximales de calcul n'atteignent pas le pétaFLOPS. Les 28 supercalculateurs suivants dans la liste GREEN 500 ont des efficacités énergétiques très voisines et dépassent à peine 2 gigaFLOPS par watt.

La question énergétique à l'échelle exaflopique devient d'autant plus préoccupante lorsque l'on sait que l'on atteint déjà des consommations énergétiques supérieures à 17 MW à l'échelle pétaflopique alors que le DARPA, le bureau de recherche et du développement du ministère de défense des Etats-Unis d'Amérique, a fixé à 20 MW le seuil de consommation à ne pas dépasser pour les machines exaflopiques [Bergman et al., 2008]. Ces systèmes qui seront plus de 30 fois plus performants que les systèmes actuels devraient donc atteindre une efficacité énergétique de 50 gigaFLOPS/W alors qu'aujourd'hui ils atteignent entre 2 et 3 gigaFLOPS/W. Par conséquent, la réduction de la consommation énergétique des infrastructures de calcul très haute performance est un enjeu majeur des prochaines années pour réussir le passage à l'ère exaflopique.

Le besoin d'aller au plus vite vers cette ère exafloplique est d'autant plus exprimé que l'on sait qu'à l'heure actuelle, certains problèmes complexes sont impossibles à

traiter dans un laps de temps raisonnable avec des machines pétaflopiques et que seule l'émergence de machines exaflopiques permettra de les résoudre. La nécessité de résoudre ces problèmes très complexes est notamment éprouvée :

- en climatologie [Hack and Bierly, 2007], pour être capable de prédire les phénomènes climatiques extrêmes tels que les canicules, sécheresses, et inondations ;
- en médecine, où la possibilité d'effectuer le calcul complexe du génome humain d'un patient rendrait possible la prescription de traitements individualisés ;
- en sismologie, pour être en mesure de prédire de façon plus détaillée les mouvements de terrain au niveau des sites exigeant une sécurité accrue et au niveau des zones où l'on s'attend fréquemment à des mouvements tectoniques [Intel, 2009] ;
- dans le domaine de l'énergie nucléaire, pour permettre d'analyser les menaces et de prédire les effets des explosions nucléaires [Kothe, 2007] ;
- mais aussi dans d'autres domaines, tels que les nanosciences [Department of Energy, 2007], la chimie, etc.

Alors que ces futures applications exaflopiques ne sont pas toutes encore conçues et développées, les services applicatifs sont déjà fortement étudiés. Nous appelons service applicatif un composant logiciel permettant de réaliser une fonctionnalité donnée pour la bonne exécution de l'application, au service de cette dernière et qui est pressenti pour être nécessaire à toutes les applications de calcul haute performance très larges échelles. En plus d'être indispensables pour la bonne exécution des applications, leurs consommations énergétiques risquent elles aussi de s'accroitre fortement lors du passage à l'échelle exaflopique. Pour ces raisons, nous nous intéressons dans cette thèse à réduire la consommation énergétique des services applicatifs plutôt que celle des applications. Plusieurs services applicatifs seront incontournables à l'échelle exaflopique :

- La fiabilité et la tolérance aux pannes : à partir des connaissances et observations actuelles sur les infrastructures existantes, il est anticipé que les systèmes exaflopiques subiront divers types de défaillances plusieurs fois par jour [Cappello et al., 2009]. Donc, afin d'exécuter des applications exaflopiques sur des systèmes distribués à très large échelle, il est indispensable de mettre en place des mécanismes de tolérance aux pannes.
- Les échanges à très large échelle de volumes de données très importants : les applications qui s'exécuteront sur ces systèmes échangeront des centaines d'exaoctets [Aloisio and Fiore, 2009]. Afin d'accroître les performances dans les applications exaflopiques, d'importantes optimisations sont nécessaires en matière d'algorithmes d'échanges collectifs de données.
- La supervision des ressources de calcul à très large échelle.
- La visualisation en temps réel des données issues de centaines de millions de noeuds, etc.

1.1 Services applicatifs considérés

Dans cette thèse, nous nous intéressons spécifiquement à deux services applicatifs : la tolérance aux pannes et la diffusion de données. D'une part, le choix de ces deux services applicatifs est motivé par le fait qu'ils font partie des services incontournables pour relever les principaux défis auxquels il est nécessaire de faire face à une échelle post-pétaflopique. D'autre part, il se justifie par une volonté d'étudier deux services dont le positionnement dans l'application est assez différent. En effet, la tolérance aux pannes est un composant logiciel qui s'exécute en arrière plan, parallèlement à l'application pendant toute la durée de son exécution. Quant à la diffusion de données, il s'agit d'un service ayant lieu à un moment donné de l'application et pouvant être assimilé à une partie de cette application. Il existe diverses façons d'implémenter chacun de ces deux services applicatifs.

1.1.1 Le service de tolérance aux pannes

Il existe trois grandes familles de protocoles de tolérance aux pannes reposant sur la sauvegarde de points de reprise : les protocoles coordonnés, les protocoles non coordonnés, et les protocoles hiérarchiques. Le principe général commun à ces trois grandes familles est de sauvegarder régulièrement (intervalle de points de reprise) l'état global de l'application afin de redémarrer cette dernière en cas de panne au dernier point de sauvegarde plutôt que de tout ré-exécuter. Le problème lorsqu'on sauvegarde les points de reprise c'est de garantir la cohérence globale du système. Un état global est dit cohérent s'il ne comporte pas de messages reçus mais non envoyés.

Les protocoles coordonnés sont actuellement les protocoles de tolérance aux pannes les plus utilisés dans les applications de calcul haute performance. Afin de garantir la cohérence globale du système, le protocole coordonné repose sur la coordination qui consiste essentiellement à synchroniser tous les processus avant chaque sauvegarde de point de reprise [Netzer and Xu, 1995]. La coordination peut avoir un impact important sur les performances car pour synchroniser tous les processus, il faut d'abord attendre que tous les messages en cours de transmission soient reçus. De plus, en cas de panne avec le protocole coordonné, tous les processus doivent redémarrer au dernier point de reprise même si un seul processus est tombé en panne. Cela conduit à un gaspillage d'énergie important car tous les processus, même ceux qui ne sont pas tombés en panne, doivent rejouer tous leurs calculs et communications déjà réalisés depuis le dernier point de reprise. La figure 1.3 schématise le fonctionnement du protocole coordonné avant et après une panne.

Le protocole non coordonné avec enregistrement de messages aborde ce problème en ne redémarrant que les processus défaillants. Dans ce cas, la consommation d'énergie de la récupération est censée être beaucoup plus faible que celle du protocole coordonné. En contrepartie, afin de garantir la cohérence globale des points de reprises sauvegardés de façon non coordonnée, les protocoles à enregistrement de messages doivent sauvegarder tous les messages envoyés par tous les processus

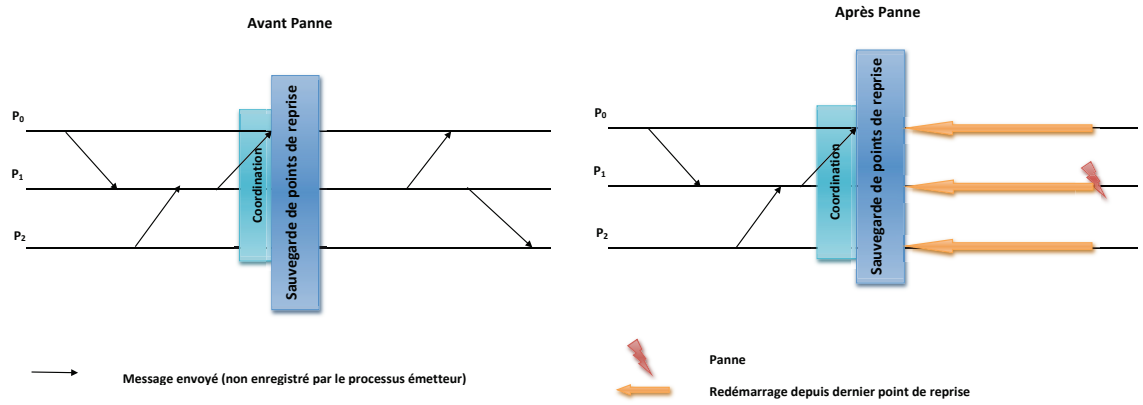

FIGURE 1.3 – Fonctionnement du protocole coordonné avant et après panne

lors de l'exécution entière, ce qui est également coûteux en performances [Bouteiller et al., 2010]. Ainsi, en cas de panne, les processus non défaillants ré-émettent aux processus défaillants les messages qu'ils ont sauvegardés. La figure 1.4 schématise le fonctionnement du protocole non coordonné avant et après une panne.

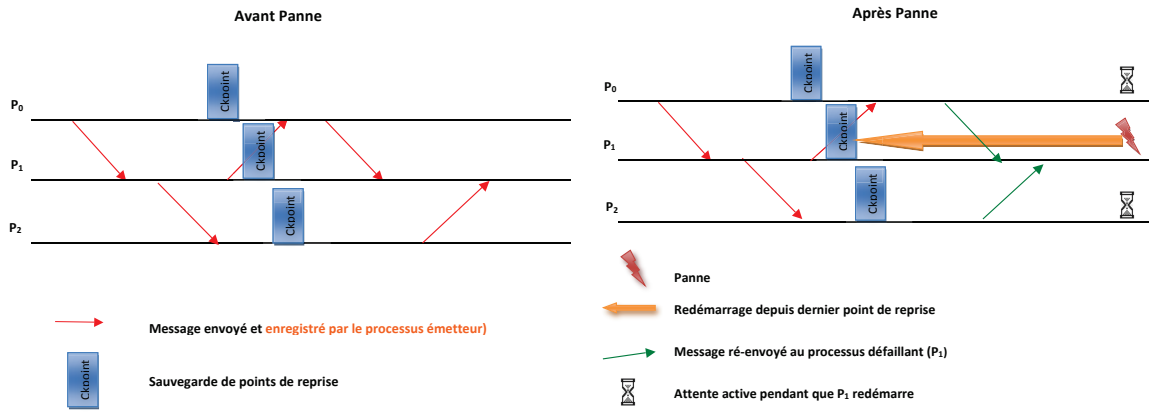

FIGURE 1.4 – Fonctionnement du protocole non coordonné avant et après panne

Une troisième famille de protocoles de tolérance aux pannes a été proposée récemment afin de remédier aux limitations des protocoles coordonnés et des protocoles à enregistrement de messages : les protocoles hiérarchiques [Guermouche et al., pear, Meneses et al., 2010, Bouteiller et al., 2011]. Ces protocoles combinent les avantages d'un enregistrement réduit de messages pendant l'exécution sans défaillance et d'un redémarrage limité en cas de récupération. Ces protocoles structurent les processus de l'exécution en grappes et enregistrent uniquement les messages échangés entre grappes. Les messages échangés au sein d'une même grappe ne sont pas sauvegardés. Ces protocoles ne mettent pas en jeu une sauvegarde coordonnée de point de reprise globale mais les processus issus d'une même grappe se coordonnent avant chaque sauvegarde de point de reprise.

Dans cette thèse, nous nous focalisons sur les deux premières familles de protocoles étant donné que la troisième famille est une combinaison de la première et de la deuxième.

1.1.2 Le service de diffusion de données

MPI (*Message Passing Interface*) est une bibliothèque de fonctions utilisables avec les langages C, C++ et Fortran permettant de faire communiquer plusieurs ordinateurs distants par passage de messages. Il s'agit d'un standard de communication incontournable pour l'exécution d'applications parallèles sur des infrastructures distribuées à très large échelle. En effet, la majorité des plates-formes pétaflopiques l'utilisent actuellement et la communauté de recherche sur le calcul très haute performance continue d'affirmer que l'on ne pourra pas s'en passer à l'échelle post-pétaflopique.

En ce qui concerne les algorithmes de diffusion de données, deux principales implémentations MPI existent : MPICH2 [8] et OpenMPI [9]. Les algorithmes utilisés diffèrent d'une implémentation à l'autre. En effet, pour diffuser un volume de données assez grand (plus de 128 Ko) à un nombre de processus assez grand (plus de 8 processus), OpenMPI utilise l'algorithme de *Pipeline* avec une taille de paquet (*chunk*) paramétrable alors que MPICH2 utilise une distribution des données partielles (*Scatter*) suivie d'une collecte généralisée des données partielles (*AllGather*). L'algorithme de distribution de données partielles de MPICH2 est implémenté en utilisant un arbre bi-nomial avec une taille de paquet égale au volume total des données divisé par le nombre total des processus. Chacune des deux implémentations (MPICH2 et OpenMPI) utilise d'autres algorithmes mais qui ne sont utilisés que pour les messages relativement petits diffusés à un nombre de processus relativement faible.

Une autre approche prometteuse pour diffuser les données est d'utiliser la programmation hybride en combinant MPI pour les communications inter-noeuds avec OpenMP pour les communications intra-noeuds [Rabenseifner et al., 2009]. En effet, MPI est optimisé pour des architectures à mémoire distribuée alors que OpenMP est plus performant pour programmer sur des mémoires partagées. Dans un algorithme hybride de diffusion de données, le processus initiateur utilise MPI pour diffuser la donnée à un seul processus MPI désigné sur chaque noeud. Ensuite, chaque processus MPI utilise OpenMP pour partager la donnée diffusée avec tous les autres processus qui se trouvent dans le même noeud.

Dans cette thèse, nous nous intéressons aux deux algorithmes de diffusion de données MPI (*Scatter & AllGather* d'une part et *Pipeline* d'autre part) ainsi qu'aux deux algorithmes de diffusion hybrides combinant OpenMP à chacun des deux algorithmes MPI.

8. MPICH2 : http://www.mcs.anl.gov/research/projects/mpich2/
9. OpenMPI : http://www.open-mpi.org/

1.2 Enjeux et problématiques

L'objectif principal de cette thèse est de réduire la consommation énergétique des infrastructures de calcul très haute performance. Comment exécuter des applications sur ces infrastructures dans l'optique de consommer moins d'énergie ? Comme expliqué précédemment, nous nous préoccupons essentiellement d'économiser l'énergie consommée par les services applicatifs (plutôt que celle des applications) dans la mesure où ils sont connus actuellement et qu'ils sont incontournables à très large échelle. Or nous venons de voir qu'il existe diverses façons d'implémenter un même service applicatif (différents protocoles de tolérance aux pannes et divers algorithmes de diffusion de données). Dans la mesure où il existe plusieurs versions (algorithmes, protocoles) pour chacun des services applicatifs, la deuxième question que l'on est amené à se poser est de savoir **quelle version choisir pour consommer moins d'énergie**. Ce choix dépend-il des paramètres d'exécution de l'application ou/et des spécificités de la plate-forme de calcul très haute performance ?

Le cas échéant, nous devons être capables de prédire la consommation énergétique des différentes versions de ces composants logiciels en fonction des paramètres d'exécution et des spécificités de la plate-forme de calcul intensif. En connaissant la consommation énergétique des différentes versions d'un service applicatif avant l'exécution de l'application, nous serons capables de choisir celui qui consomme le moins d'énergie. **Comment estimer de façon précise avant l'exécution de l'application, la consommation énergétique des différentes versions d'un même service applicatif ?** Pour répondre à cette question, il est d'une part essentiel de savoir comment tenir compte des paramètres d'exécution. D'autre part, il est important de mettre en place une méthodologie afin d'adapter les modèles théoriques d'estimation énergétique aux caractéristiques architecturales (processeurs, coeurs de calcul, mémoires, réseaux, etc.), matérielles (types de supports de stockage, technologie réseau, etc), logicielles (système d'exploitation) et environnementales (température de la salle d'expérimentation) de la plate-forme expérimentale. Les modèles théoriques d'estimation énergétique sont les équations permettant de calculer l'énergie consommée à partir de tous les paramètres faisant varier la consommation énergétique.

Pour adapter les estimations énergétiques au matériel d'exécution, nous avons besoin de procéder à un ensemble de mesures de puissance électrique et de temps d'exécution permettant de calibrer les modèles d'estimation en fonction de l'infrastructure de calcul intensif. Nous appelons calibration le processus qui consiste à orchestrer la prise de ces mesures de puissance électrique et de temps d'exécution. Afin de calibrer les modèles d'estimation sur une plate-forme donnée, il faut être capable de mesurer la consommation électrique pendant l'exécution des services applicatifs. **Comment mesurer la consommation électrique des services applicatifs ?** Comment s'assurer de la précision de la valeur relevée à l'aide d'un dispositif de mesure de la consommation électrique ? Quelle fréquence de mesure est nécessaire

pour tenir compte des éventuelles variations électriques ? Par ailleurs, une infrastructure de calcul intensif comporte une ou plusieurs grappes de calcul constituées de noeuds identiques d'un point de vue matériel et logiciel. Comment se comportent ces noeuds identiques vis à vis de la consommation électrique ? Ont-ils tous la même consommation s'ils exécutent les mêmes applications ?

En plus d'être très importante, la consommation électrique des supercalculateurs est dynamique. En effet, la dynamique de consommation électrique est une conséquence d'une utilisation irrégulière des ressources de calcul et d'un plafonnement variable de la puissance électrique totale que peut délivrer le fournisseur d'électricité. Par conséquent, nous pensons qu'il est indispensable de mettre en place un dialogue avec les différents acteurs qui interagissent avec l'infrastructure de calcul intensif. En plus de l'utilisateur qui souhaite exécuter ses applications et l'administrateur qui configure et supervise la plate-forme expérimentale, le fournisseur d'énergie joue également un rôle très important dans la mesure où il permet l'alimentation électrique de l'infrastructure. Selon son approvisionnement en énergie et des tendances de consommation de la société, l'électricité qu'il fournit peut être proposée à des tarifs financiers fluctuants et est à l'origine d'une pollution variable de l'environnement quantifiée en grammes de CO_2 émis.

À cet effet, l'usage d'une grille électrique intelligente permet de gérer le dialogue entre le fournisseur d'énergie et le consommateur qui, dans notre cas est l'infrastructure de calcul très haute performance. **Un autre objectif de cette thèse est de "consommer mieux"** en s'appuyant sur cette grille électrique intelligente mais également sur la calibration électrique de la plate-forme de calcul intensif. "Consommer mieux" signifie que nous consommons la même quantité d'énergie mais à moindres coûts financiers tout en générant moins de pollution de l'environnement. Comment tenir compte des contraintes des différents acteurs afin de "consommer mieux" selon plusieurs critères (consommation d'énergie, coûts financier et environnemental) ? Pour cela, il faut veiller à planifier les réservations des ressources de calcul de sorte que l'on puisse consommer "moins" et "mieux".

1.3 Contributions et annonce du plan

Dans la suite du présent manuscrit, nous allons chercher à répondre aux questions précédemment énoncées. Pour cela, nous nous appuyons essentiellement sur des expérimentations pour valider nos réponses.

Le chapitre 2 présente un état de l'art des différentes solutions pour mesurer, estimer et réduire la consommation énergétique des serveurs de calcul. Il présente également les différentes solutions pour planifier des réservations de ressources de calcul dans une plate-forme distribuée tout en optimisant la consommation énergétique.

Dans le chapitre 3, nous évaluons et analysons la consommation électrique de noeuds de calcul haute performance à partir de mesures réelles issues de différents scénarii. Le premier objectif de ce chapitre est de souligner les difficultés pour me-

surer avec précision l'énergie consommée par un service applicatif donné. Ces difficultés sont mises en avant grâce à l'étude des divergences entre les mesures issues de différents appareils de mesures (externes/internes, avec une fréquence de mesure faible/élevée). Le deuxième objectif de ce chapitre est d'étudier le comportement de la consommation électrique de noeuds identiques.

Dans le chapitre 4, nous présentons notre approche d'estimation de la consommation d'énergie pour les protocoles de tolérance aux pannes et les algorithmes de diffusion de données. Nous présentons les modèles selon lesquels nous évaluons l'énergie consommée par les différents services étudiés. Bien que génériques, ces modèles ne suffisent pas pour estimer de façon précise la consommation énergétique des différentes versions d'un service donné. En effet, la consommation énergétique d'un service applicatif est très dépendante du matériel utilisé dans la plate-forme d'exécution. Afin d'adapter nos modèles à une plate-forme donnée, notre approche d'estimation repose donc sur une calibration de la plate-forme basée sur des mesures énergétiques réelles des différentes opérations de base identifiées.

Le chapitre 5 présente les différentes solutions pour réduire la consommation d'énergie dans une plate-forme de calcul très haute performance. Dans un premier temps, nous étudions la possibilité de réduire la consommation énergétique en nous basant sur les résultats de calibration électrique des noeuds de calcul qui constituent l'infrastructure de calcul très haute performance. Dans un deuxième temps, nous mettons en évidence comment il est possible de réduire la consommation énergétique des applications en nous appuyant sur les estimations énergétiques des différentes versions de services applicatifs.

Le chapitre 6 présente SESAMES, un environnement logiciel qui intègre des calibrations et estimations énergétiques que nous sommes capables de fournir. Dans ce chapitre sont présentées les différentes fonctionnalités de SESAMES. Nous décrivons les différents composants ainsi que les interactions de SESAMES avec les multiples acteurs de la plate-forme de calcul très haute performance. Nous décrivons en particulier les échanges SESAMES/utilisateurs et les échanges SESAMES/fournisseur d'énergie qui ont lieu dans le cadre d'une grille électrique intelligente. En nous appuyant sur SESAMES, nous présentons notre approche efficace en énergie et multi-objectifs (moins d'énergie, moins de coûts financiers et environnementaux) pour planifier les réservations de ressources qu'expriment les différents utilisateurs de la plate-forme de calcul très haute performance.

Le chapitre 7 présente les conclusions de cette thèse et ouvre de nouvelles perspectives et voies de recherche.

Etat de l'art

2

La question de l'efficacité énergétique dans les infrastructures de calcul très haute performance est de plus en plus abordée dans les travaux de recherche scientifique, dans la mesure où la consommation énergétique de ces plates-formes constitue un enjeu majeur. Dans ce chapitre, nous nous intéressons à montrer comment ces différents travaux ont cherché à répondre à quelques problématiques que nous avons identifiées dans le chapitre précédent.

La section 2.1 présente les différents travaux qui été proposés afin d'évaluer la consommation énergétique lors de l'exécution des applications, que ce soit par la mesure directe ou par l'estimation. La section 2.2 présente les différentes solutions proposées afin de consommer l'énergie de façon réduite et meilleure.

2.1 Évaluation de la consommation énergétique lors de l'exécution d'applications

Pour évaluer la consommation énergétique d'applications, la première approche consiste à utiliser des dispositifs qui mesurent la puissance électrique alimentant les différentes machines sur lesquelles est exécutée une application. La deuxième approche consiste à estimer la consommation énergétique en s'appuyant sur des modèles théoriques, généralement liés aux performances. La première méthode est souvent utilisée pour évaluer la précision des approches d'estimation.

2.1.1 Mesure de la puissance électrique

Pour mesurer la consommation énergétique des systèmes informatiques, deux méthodes sont principalement utilisées : soit on utilise des équipements matériels que l'on connecte à l'intérieur ou à l'extérieur de chaque machine, soit on utilise des capteurs d'énergie qui sont intégrés à certaines architectures matérielles.

2.1.1.1 Quel est le bon équipement de mesure électrique ?

Plusieurs travaux de recherche se sont intéressés à la mesure de la puissance électrique des serveurs ou des systèmes distribués à large échelle. Pour cela, différents dispositifs ont été utilisés.

Dans l'article [Carpentier et al., 2012], les auteurs supervisent la consommation énergétique de serveurs à l'aide de quatre différents dispositifs de mesure de la puissance électrique :

- le PDU [1] Eaton [2] ;
- le PDU Schleifenbauer [3] ;
- le wattmètre OmegaWatt [4] ;
- une carte de supervision proposée par plusieurs constructeurs (Dell, Intel, HP et Nec) : l'interface de gestion intelligente de matériel [Intel et al., 2009] (IPMI [5]) est un ensemble de spécifications d'interfaces permettant de superviser la consommation énergétique et la température à l'intérieur des machines.

Pour mesurer la consommation énergétique, d'autres dispositifs assez similaires sont également utilisés :

- le wattmètre WattsUp utilisé dans [Castillo et al., 2011] ;
- la carte de supervision PowerExecutive proposée dans les serveurs IBM BladeCenter [6] ;

Ces dispositifs de mesure se différencient notamment par quatre critères : la nature, la localité, la précision et la fréquence de mesure. OmegaWatt et WattsUp mesurent la puissance active, c'est-à-dire la puissance électrique moyenne calculée à partir de l'énergie mesurée pendant une seconde alors que les autres dispositifs mesurent une puissance électrique instantanée qui est obtenue en mesurant le courant et la tension aux bornes de la machine. IPMI (utilisé aussi dans [Giri and Vanchi, 2010]) et PowerExecutive permettent de mesurer la puissance électrique à l'intérieur de la machine (à la sortie du bloc d'alimentation) contrairement aux autres dispositifs qui la mesurent en externe (à l'entrée du bloc d'alimentation). Cela signifie que la mesure électrique fournie par IPMI est en courant continu et ne tient pas compte de la puissance électrique dissipée par le bloc d'alimentation du serveur. À l'opposé, les trois autres dispositifs mesurent en courant alternatif la puissance électrique globale alimentant la machine et tient donc compte du bloc d'alimentation et de tous les composants matériels de la machine. Ainsi, un inconvénient de IPMI et PowerExecutive est que ces dispositifs ne tiennent pas compte de la puissance électrique dissipée par le bloc d'alimentation du serveur qui peut représenter une proportion importante de la consommation totale du serveur (environ 20 %).

1. PDU : Power Distribution Unit ou unité de distribution d'électricité
2. Eaton : http://www.eaton.fr
3. Schleifenbauer : www.schleifenbauer.eu
4. Omegawatt : http://www.omegawatt.fr
5. IPMI : Intelligent Platform Management Interface
6. IBM PowerExecutive : http://www-03.ibm.com/systems/management/director/about/director52/extensions/powerexec.html

Bien que IPMI et PowerExecutive présentent l'avantage de superviser la puissance électrique sans avoir à installer un équipement supplémentaire, cette solution s'avère être peu précise et ne permet pas d'avoir une granularité de mesure assez fine (puissance instantanée à une fréquence relativement faible). En effet, en ce qui concerne IPMI, une incertitude de mesure de $\pm 7W$ est trop grande pour mesurer des fluctuations électriques de quelques watts alors que les wattmètres OmegaWatt, WattsUp et le PDU Schleifenbauer affichent une précision de l'ordre de $\pm 0.1W$. Le PDU Eaton a une précision intermédiaire de $\pm 1W$. De plus, les deux cartes de supervision (IPMI et PowerExecutive) présentent une fréquence de mesure assez faible (1 mesure toutes les 5 secondes pour IPMI). Bien que leurs précisions de mesure soient relativement élevées, les autres dispositifs ne fournissent qu'une mesure par seconde au meilleur des cas. En fournissant des puissances électriques chaque 3 secondes, les PDUs Schleifenbauer et Eaton ne sont donc pas adaptés pour mesurer la consommation énergétique des applications ou des tâches qui ne durent que quelques secondes. À l'opposé, OmegaWatt et WattsUp fournissent une mesure qui tient donc compte des fluctuations électriques pendant une seconde étant donné qu'ils mesurent une puissance moyenne chaque seconde.

Pour illustrer cela, la figure 2.1 compare la puissance électrique totale d'une grappe de six noeuds mesurée simultanément par IPMI et le PDU Eaton. L'axe des abscisses affiche l'horaire de la mesure tandis que l'axe des ordonnées présente la puissance électrique totale en watts. La courbe représente l'évolution de puissance électrique totale mesurée avec IPMI alors que les histogrammes représentent celle du PDU Eaton. Même si la puissance électrique mesurée par IPMI est censée être plus basse (car ne tient pas compte des blocs d'alimentation) que celle qui est relevée par le PDU Eaton, nous observons un scénario différent entre 15h55 et 16h15 et ce à cause de la précision des mesures qui est beaucoup plus faible avec IPMI ($\pm 7W$ *vs* $\pm 1W$) sur chaque noeud. Par ailleurs, nous observons qu'il y a un décalage temporel entre les variations électriques mesurées par IPMI et celles qui sont relevées par le PDU Eaton. Les mesures prises par IPMI sont en retard par rapport à celles mesurées avec le PDU Eaton (voir 15h52). Ce décalage temporel est lié à la fréquence de mesure de IPMI qui est relativement plus basse que celle du PDU Eaton (1 mesure toutes les 5 secondes *vs* 1 mesure chaque 3 secondes).

Il existe d'autres équipements mesurant à des fréquences plus élevées (supérieures à 1 Hz). En effet, des équipements de mesure interne comme PowerMon2 [Bedard et al., 2010] permettent de mesurer à des fréquences très élevées la puissance électrique interne instantanée des différentes lignes électriques ATX (3.3 V, 5 V et 12 V) qui sortent du bloc d'alimentation de la machine et qui alimentent la carte mère et les différents périphériques. Pour mesurer la puissance électrique instantanée d'une ligne électrique, PowerMon2 mesure la valeur du courant et la multiplie par la tension qui est constante (3 V, 5 V ou 12 V). Lorsque PowerMon2 est utilisé pour mesurer la puissance électrique d'une seule ligne électrique, sa fréquence de mesure peut atteindre 3072 Hz.

Les auteurs de l'article [Barrachina et al., 2013] ont utilisé trois autres équipe-

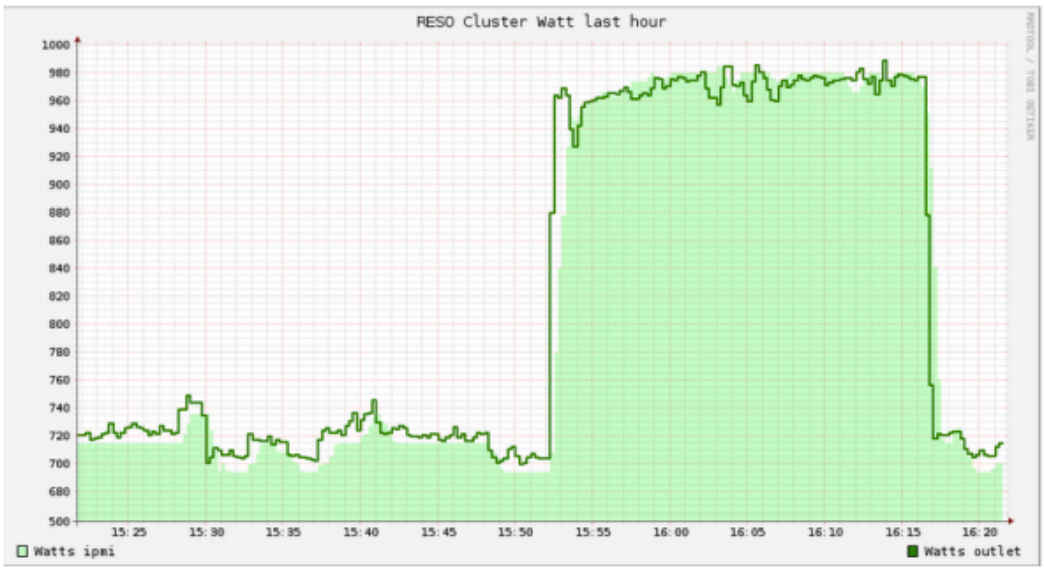

FIGURE 2.1 – Mesures de la puissance électrique à l'aide du PDU Eaton et de IPMI (figure extraite de [Carpentier et al., 2012])

ments de mesure interne très similaires à POWERMON2. Parmi eux, ils ont utilisé NI qui peut atteindre 7000 Hz s'il mesure la puissance électrique d'une seule ligne électrique. Les deux autres équipements de mesure interne DCM et DC2M ont été fabriqués par leurs soins et permettent de mesurer la puissance électrique instantanée à des fréquences respectivement égales à 25 Hz et 100 Hz. Contrairement à POWERMON2, ces trois équipements (NI, DCM et DC2M) ne permettent de mesurer que les lignes électriques de 12V qui permettent d'alimenter les processeurs, les ventilateurs et une partie de la carte mère. Par exemple, ils ne permettent pas de mesurer la puissance électrique alimentant certains périphériques tel que le disque dur.

L'inconvénient de ces équipements de mesure interne est qu'ils ne sont pas faciles à brancher et qu'ils ne conviennent que pour des blocs d'alimentation standards, c'est-à-dire avec des câbles électriques respectant la norme ATX. Leur déploiement est donc inapplicable à très large échelle.

Les auteurs des articles [Hähnel et al., 2012, Rotem et al., 2012] utilisent les compteurs d'énergie RAPL [7] disponibles uniquement dans les processeurs Intel Sandy et Ivy Bridge. D'autres compteurs d'énergie équivalents existent sur les processeurs AMD qui sont par exemple utilisés dans le supercalculateur BlueWaters [8]. Les compteurs d'énergie RAPL permettent de mesurer la puissance électrique instantanée de ces processeurs Intel à une fréquence égale à 1000 Hz et avec une précision assez élevée (inférieure à 1 W). Ils sont intéressants pour superviser la puissance électrique

7. RAPL : Running Average Power Limit
8. Projet BlueWaters : https://bluewaters.ncsa.illinois.edu

d'applications s'exécutant sur des systèmes distribués à très large échelle dans la mesure où ils ne nécessitent aucun branchement et mesurent la puissance instantanée avec une précision et une fréquence élevées. Cependant, ils ne mesurent que la puissance électrique des processeurs (et non pas de toute la machine). D'autre part, ils ne sont disponibles que dans les familles de processeurs Intel Sandy et Ivy Bridge. De plus, il s'agit d'une solution de mesure intrusive dans la mesure où il faut accéder souvent aux compteurs d'énergie pour lire les valeurs de puissance électrique. Pour cela, il est nécessaire d'exécuter un programme qui collecte les mesures. Il faut donc s'assurer que l'utilisateur a les droits suffisants pour exécuter ce programme. L'exécution d'un tel programme peut fausser la mesure de la puissance électrique.

Ainsi, nous pouvons classer tous ces dispositifs de mesure en cinq catégories présentées dans le tableau 2.1.

Catégorie	Puissance	Localité	Précision	Fréquence
PDUs (Eaton et Shleifenbauer)	moyenne	externe	intermédiaire	faible
Wattmètres externes (OmegaWatt et WattsUp)	moyenne	externe	élevée	intermédiaire
Wattmètres internes (PowerMon2, NI, DCM et DC2M)	instantannée	interne	élevée	élevée
Compteurs d'énergie (RAPL)	instantannée	interne	élevée	élevée
Cartes de supervision (IPMI, PowerExecutive)	instantannée	interne	faible	faible

Table 2.1 – Classification des différents dispositifs de mesure de la puissance électrique

Les wattmètres externes et les PDUs mesurent et affichent la puissance électrique active (ou moyenne) qui est obtenue en divisant par une période de temps T (liée à leur fréquence de mesure) l'énergie consommée pendant T. L'énergie consommée est approximée par une intégration sur un nombre de mesures de la puissance instantanée plus ou moins élevé. Les wattmètres internes mesurent les puissances électriques instantanées des lignes ATX servant à alimenter les différents composants de la machine. Étant donné que ces wattmètres mesurent un courant continu et que la tension aux bornes de chaque ligne ATX est connue (3.3 V, 5 V ou 12 V), ces wattmètres n'ont besoin de mesurer que le courant électrique qui alimente chacune des lignes. La puissance instantanée est obtenue en multipliant le courant par la tension aux bornes de la ligne ATX. En ce qui concerne les compteurs d'énergie (RAPL) et les cartes de supervision (IPMI et PowerExecutive), nous n'avons pas d'informations relatives aux principe de la mesure.

Ces travaux de recherche **ne permettent pas de savoir s'il convient mieux d'utiliser une catégorie de dispositifs de mesure plutôt qu'une autre en fonction de l'usage que l'on souhaite en faire**. Dans notre étude, nous nous intéresserons à comparer les wattmètres externes et les wattmètres internes pour savoir lesquels sont les plus appropriés pour nos mesures de la puissance électrique des services applicatifs.

2.1.1.2 Comment collecter les mesures de puissance électrique à grande échelle et en déduire la consommation de services applicatifs ?

Un des premiers travaux qui utilise des dispositifs de mesure de la puissance électrique est effectué par Viredaz et al. sur des systèmes informatiques mobiles [Viredaz and Wallach, 2000]. Joseph et al. utilise une méthode similaire pour mesurer la puissance électrique d'un processeur de haute performance [Joseph and Martonosi, 2001, Joseph et al., 2001]. Pour déduire la consommation énergétique, ces premiers travaux utilisent un ensemble de bancs d'essai pour extraire le temps d'exécution et la puissance électrique dissipée. Dans [Kamil et al., 2008], les auteurs mesurent la puissance électrique d'un superordinateur Cray XT4 lors de différentes exécutions d'applications de calcul haute performance. Pour obtenir leurs mesures, ils additionnent les puissances électriques affichées pour sur le panneau de contrôle de chaque armoire de noeuds du supercalculateur. Leurs mesures montrent que les bancs d'essai effectuant des calculs intensifs génèrent la dissipation de puissance électrique la plus élevée.

Pour superviser la puissance électrique des noeuds d'un système distribué à large échelle, les auteurs des articles [Dias de Assuncao et al., 2010b, Dias de Assuncao et al., 2010a] utilisent une infrastructure de wattmètres OmegaWatt pour mesurer la consommation énergétique de chacun des noeuds du site lyonnais de la plateforme expérimentale Grid5000 [Cappello et al., 2005]. Ces wattmètres mesurent périodiquement l'énergie consommée par 135 noeuds et un routeur. Quatre boîtes Omegawatt ont été utilisées : une boîte de 48 ports, deux boîtes de 42 ports et une boîte de 6 ports. Chaque boîte fonctionne comme un appareil de mesure multi-prises.

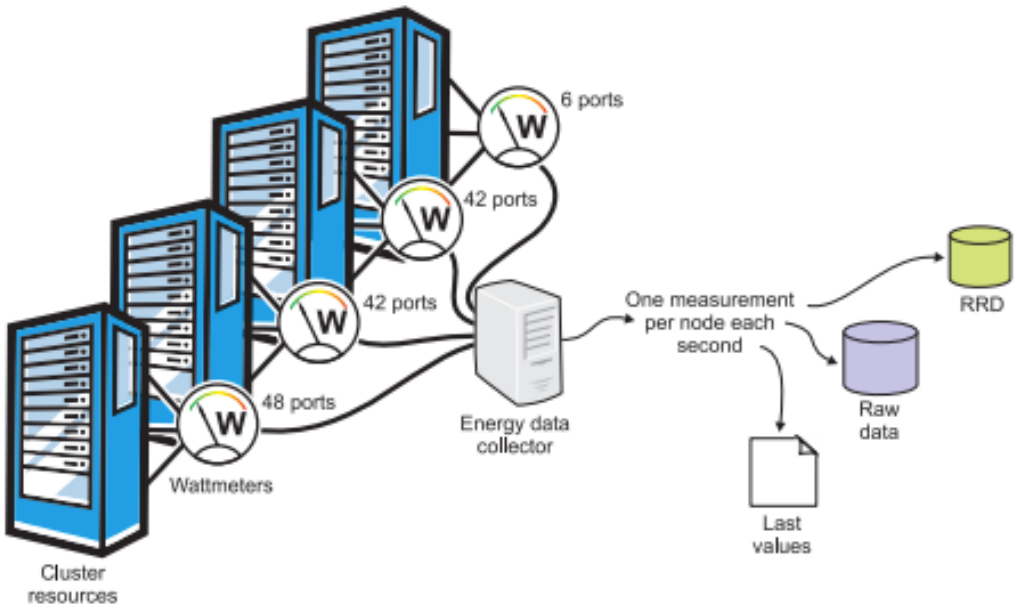

FIGURE 2.2 – Dispositif expérimental de mesure avec des wattmètres OMEGAWATT (figure extraite de [Dias de Assuncao et al., 2010b])

Les données d'énergie sont transmises à un serveur collecteur de données qui est connecté aux quatre wattmètres OMEGAWATT par quatre liaisons série distinctes. Sur le serveur collecteur de données, un processus démon permet de collecter les données d'énergie pour chaque boîte OMEGAWATT. À chaque seconde, ce démon envoie une requête de données dans un code hexadécimal avec un CRC à la boîte pour demander la valeur de puissance électrique de chacune des ressources contrôlées par cette boîte. La réponse est un code hexadécimal avec un CRC sous la forme de "paquets", où chaque paquet comporte les puissances électriques moyennes de six noeuds.

Les données énergétiques sont stockées de trois façons différentes :

- Données brutes ("Raw values") : un fichier par ressource supervisée qui comprend également un horodatage indiquant l'instant de la mesure.
- Dernières valeurs ("Last values") : un fichier par ressource où la valeur de puissance électrique est écrasée à chaque seconde.
- Bases de données round-robin [9].

La collecte et le stockage de données d'énergie pour le site lyonnais de Grid'5000 sont illustrés sur la figure 2.2. Les mesures stockées peuvent être obtenues par les utilisateurs sur le site web https://helpdesk.grid5000.fr/supervision/Lyon/wattmetre. Pour cela, il suffit de spécifier les instants du début et de fin de la mesure ainsi que les noeuds pour lesquels l'utilisateur souhaite obtenir les mesures.

Bien que cette approche soit conviviale pour l'utilisateur, elle n'est pas adéquate

9. RRD tool http://oss.oetiker.ch/rrdtool

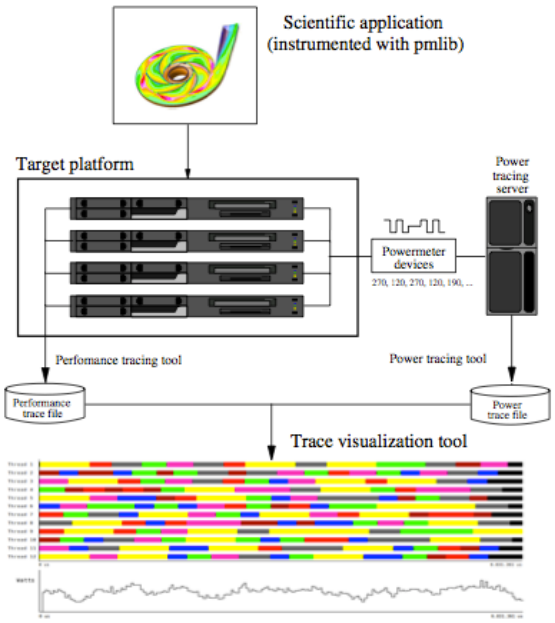

FIGURE 2.3 – Librairie `pmlib` pour relever les mesures de puissance électrique (figure extraite de [Barrachina et al., 2013])

pour mesurer la puissance électrique d'une application s'exécutant sur des noeuds de la plate-forme dans la mesure où l'horodatage des mesures est effectué par le serveur collecteur de données pour lequel l'horloge n'est pas forcément synchronisée avec celles des noeuds supervisés. Ainsi, la première mesure qui sera collectée par l'utilisateur ne concordera pas forcément avec le début de l'application. Pour pallier ce problème, il est indispensable de synchroniser les horloges des noeuds supervisés avec celle du serveur collecteur de mesures ou de récupérer les dates de début et de fin de l'application à partir de ce serveur.

Les auteurs des articles [Barrachina et al., 2013, Alonso et al., 2012] proposent une librairie `pmlib` permettant de superviser la consommation énergétique de façon plus corrélée à l'exécution d'une application de calcul haute performance. Ce paquet logiciel est développé et maintenu par l'équipe de recherche HPC&A de l'Université Jaume I. La figure 2.3 présente un schéma global de l'architecture utilisée.

Une application de calcul haute performance s'exécute sur une plate-forme distribuée sur laquelle sont connectés un ou plusieurs dispositifs de mesure de la puissance électrique (wattmètres externes et internes). Pour relever les mesures de puissance électrique de façon corrélée à l'application, la librairie pmlib utilise un ensemble de routines permettant notamment à l'utilisateur de démarrer et d'arrêter la collecte de mesures. Pour cela, il instrumente son application pour placer les routines de pmlib en fonction de ce qu'il souhaite mesurer. Les mesures prélevées sont stockées sur le serveur collecteur de données énergétiques et fournies à l'utilisateur à la fin de l'exécution de son application. Ces mesures énergétiques peuvent également être

visualisées à l'aide de l'outil Paraver [10].

L'environnement logiciel PowerPack [Ge et al., 2009] s'appuie sur les wattmètres NI pour superviser la consommation énergétique d'un noeud d'une plate-forme de calcul haute performance. En s'appuyant sur les mesures énergétiques d'un seul noeud d'une grappe homogène, les auteurs émulent la mesure de la consommation énergétique sur plusieurs noeuds de cette grappe. Leur approche suppose donc que tous les noeuds de la grappe consomment la même puissance électrique. Nous montrons dans notre thèse que ceci n'est pas vrai et n'est pas applicable.

Bien que de nombreux travaux de recherche aient utilisé les différents dispositifs pour mesurer la consommation énergétique des systèmes informatiques ou au mieux celle des applications s'exécutant sur ces systèmes, **aucun d'eux ne s'est intéressé à mesurer la consommation énergétique de certaines opérations spécifiques dans des services applicatifs tels que les protocoles de tolérance aux pannes ou les algorithmes de diffusion de données**. Généralement, les travaux qui s'intéressent à ces deux services applicatifs ne cherchent à mesurer que les performances des différents protocoles de tolérance aux pannes [Ndiaye et al., 2012, Guermouche et al., 2011, Ropars et al., 2011, Moody et al., 2010, Bouguerra et al., 2010, Bougeret et al., 2011, Ho et al., 2008] ou les algorithmes de diffusion de données [Wadsworth and Chen, 2008, Pjesivac-Grbovic et al., 2005].

2.1.2 Estimation de la consommation énergétique

Même si la mesure directe de la consommation énergétique est plus précise que l'estimation de cette consommation, le coût du déploiement d'équipements matériels rend cette mesure très onéreuse voire inapplicable à très larges échelles. Par conséquent, plusieurs travaux ont cherché à estimer la consommation énergétique en établissant des modèles. Les modèles de consommation énergétique ont été proposés soit au niveau d'un composant matériel, soit à un niveau plus global (noeud, système, application). Estiment-ils avec précision la consommation énergétique ? Permettent-ils de fournir cette estimation énergétique avant d'exécuter l'application ?

2.1.2.1 Au niveau d'un composant matériel

La plupart des précédents travaux de recherche se sont focalisés sur l'estimation de la consommation énergétique d'un seul composant matériel :

- le processeur : Wattch [Brooks et al., 2000], SimplePower [Ye et al., 2000] et SoftWatt [Gurumurthi et al., 2002] ;
- la mémoire : SimplePower [Ye et al., 2000] et SoftWatt [Gurumurthi et al., 2002] ;
- le disque dur : Dempsy [Zedlewski et al., 2003] et [Gurumurthi et al., 2003, Molaro et al., 2009] ;
- les interfaces réseaux : Orion [Wang et al., 2002] et [Ye et al., 2002].

Pour estimer la consommation énergétique, les auteurs des articles [Brooks et al., 2000, Gurumurthi et al., 2003, Janzen, 2001, Wang et al., 2002, Ye et al., 2000, Zed-

10. Paraver : http://www.cepba.upc.es/paraver

lewski et al., 2003] simulent le comportement des différentes ressources pour lesquelles ils souhaitent connaître la consommation en s'appuyant sur des modèles théoriques. Pour cela, les travaux comme [Brooks et al., 2000, Ye et al., 2000] modélisent la puissance électrique de différents composants matériels à partir d'une analyse de ces composants au niveau microarchitectural, c'est-à-dire en se basant sur la cartographie du processeur au niveau logique : nombre de mémoire cache, nombre et longueur des pipelines, les différents registres, etc. L'inconvénient de ces approches est que ce genre d'informations n'est pas toujours disponible et qu'elles sont de plus en plus difficiles à mettre en oeuvre avec les processeurs actuels qui se caractérisent par une microarchitecture souvent très complexe. Par ailleurs, ces techniques sont limitées à l'estimation de la consommation énergétique d'un seul composant du noeud et ne permettent donc pas de connaître la consommation énergétique de l'ensemble du système.

Pour contourner ce problème, d'autres approches tel que [Isci and Martonosi, 2003, Lim et al., 2010] estiment la consommation énergétique de ces composants matériels en s'appuyant sur les compteurs internes de performance (différents des compteurs d'énergie qui donnent directement la mesure énergétique). SoftPower [Lim et al., 2010] propose d'estimer la consommation énergétique du processeur, de la mémoire et de toute la machine en s'appuyant également sur les compteurs internes de performances. Leur méthodologie d'estimation a été évaluée sur une machine dotée d'un processeur Intel Core i7. La plupart des compteurs specifiques aux processeurs Intel Core i7 ont été utilisés. Par exemple, ils ont utilisé entre autres, le nombre de cycles ininterrompus du CPU (UNHALTED_CORE_CYCLES) et le nombre d'instructions terminées (INSTRUCTIONS_RETIRED) pour estimer la consommation énergétique du processeur. La figure 2.4 présente la liste des compteurs matériels de performance utilisés pour estimer la consommation énergétique de chaque ressource.

L'inconvénient de ces approches est que les compteurs de performance utilisés ne sont pas standardisés et sont spécifiques à chaque famille de processeurs. Les modèles utilisés pour estimer la consommation énergétique doivent donc être adaptés à chaque famille de processeurs. De plus, elles sont intrusives dans la mesure où il faut exécuter un programme accéder souvent aux compteurs internes de performance et lire les valeurs des différentes variables. L'exécution d'un tel programme peut fausser les valeurs des variables lues sur les compteurs de performance. En outre, ces compteurs de performance ne tiennent pas compte de la variabilité de la puissance électrique des processeurs [McCullough et al., 2011]. Toutes ces approches considèrent que la consommation énergétique est uniquement corrélée aux performances du composant matériel. **Elles ne fournissent donc pas des estimations assez précises** aussi car elles ne tiennent pas compte des autres facteurs qui peuvent influencer la consommation énergétique tels que la température et la localisation de la machine dans la salle d'expérimentation [Varsamopoulos et al., 2009, Tang et al., 2007].

Power source	Performance Event
CPU	UNHALTED_CORE_CYCLES **UNC_LLC_HITS:ANY** L1D_ALL_REF:ANY UOPS_RETIRED:ANY INSTRUCTIONS_RETIRED
Memory	**UNC_QHL_REQUESTS:** **LOCAL_READS** UNC_GQ_DATA:TO_LLC UNC_LLC_LINES_IN:ANY UNC_LLC_MISS:ANY UNC_LLC_LINES_OUT:ANY
Total	**RESOURCE_STALLS:ANY** UNHALTED_CORE_CYCLES UNC_LLC_HITS:ANY **L1D_ALL_REF:ANY** UOPS_RETIRED:ANY

FIGURE 2.4 – Compteurs de performance utilisés pour l'estimation de chaque ressource (Figure extraite de [Lim et al., 2010])

2.1.2.2 Au niveau d'une machine ou d'une application

Dans l'article [Da Costa and Hlavacs, 2010], les auteurs estiment la consommation énergétique d'un ordinateur par une méthodologie qui ne nécessite de connaître ni l'architecture de la machine, ni les compteurs internes de performance liés aux composants matériels. Pour évaluer la consommation énergétique de la machine, les auteurs essaient d'établir des modèles qui utilisent les valeurs des variables stockées dans les compteurs logiciels de performance fournis par Linux. Leur méthodologie est basée sur une fouille de données qui consiste à évaluer plus de 160 variables liées aux performances à partir de 10000 points de mesure.

Afin de déterminer les différentes variables, ils ont exécuté sur un ordinateur de bureau, des bancs d'essais qui utilisent intensément et spécifiquement certains composants de la machine (par exemple, `cpuburn` pour utiliser intensément le processeur). Ensuite, ils se sont appuyés sur la méthode des moindres carrés [Rao et al., 1999] pour établir des modèles d'estimation de la consommation énergétique de la machine à partir des variables déterminées. Bien qu'elle fournisse des estimations assez précises, cette approche ne permet pas de connaître la consommation énergétique d'une application avant de l'exécuter et son caractère intrusif lié à la lecture des compteurs logiciels de performance peut fausser les évaluations énergétiques qu'elle fournit.

Développée par l'université de Wayne, pTop [Do et al., 2009] est une librairie de supervision énergétique où le calcul énergétique se base sur les modèles de consommation des différents composants matériels (CPU, mémoire, disque, etc.).

Par exemple, pour estimer la consommation énergétique du disque dur qui est liée au processus i, il s'appuie sur le modèle suivant :

$$E_{Disk_i} = t_{read_i} \cdot P_{read} + t_{write_i} \cdot P_{write}$$

t_{read_i} et t_{write_i} sont les temps passés par le processus i respectivement pour lire et écrire des données sur le disque dur. P_{read} et P_{write} sont les puissances électriques du disque lorsqu'il effectue des accès en lecture ou en écriture. Dans leur approche, ils considèrent que les valeurs de ces puissances électriques sont récupérées des spécifications matérielles fournies par le constructeur du matériel.

Cependant, cet outil n'offre pas de support de configuration : pour adapter cet outil au matériel utilisé, il faut modifier le code source donc cela nécessite de recompiler à chaque fois l'outil après chaque nouvelle configuration. Microsoft a également proposé JouleMeter [11], un outil de supervision énergétique des composants matériels, qui permet de fournir à la fois la consommation énergétique d'un composant matériel et celle d'une application. Pour son estimation énergétique, il se base sur des informations tels que le taux d'utilisation du CPUs et la luminosité de l'écran. Toutefois, cet outil est exclusivement développé pour les plates-formes Windows.

Dans [Bourdon et al., 2013], les auteurs proposent PowerAPI, une librairie offrant une API qui permet d'estimer la consommation énergétique d'une application ou d'un processus système. L'évaluation se base également sur des modèles de consommation énergétiques des composants matériels (CPU, mémoire, disque, etc.) utilisés par le processus. Par exemple, pour le CPU, le modèle utilisé est :

$$P_{CPU}(f, V) = c \cdot f \cdot V^2$$

f est la fréquence du CPU, V est la tension à ses bornes et c une valeur qui dépend des spécifications du processeur (comme la capacitance).

La figure 2.5 présente l'architecture de PowerAPI.

L'architecture de PowerAPI est modulaire : chaque module représente une unité de calcul propre permettant d'évaluer la consommation énergétique d'une ressource matérielle. L'utilisateur peut ainsi adapter la librairie s'il connaît les spécificités techniques des composants matériels supervisés. L'utilisateur a donc besoin de se procurer ces spécifications sur les différentes documentations fournies par les constructeurs pour paramètrer cet outil.

L'un des rares travaux qui s'intéresse à l'évaluation de la consommation énergétique des mécanismes de tolérance aux pannes est l'article [Meneses et al., 2012]. Dans ce papier, les auteurs ont présenté des modèles pour évaluer à très large échelle, la consommation énergétique de trois opérations utilisées dans les protocoles de tolérance aux pannes : la sauvegarde de points de reprise, l'enregistrement de messages et la récupération de points de reprise.

11. JouleMeter : http://research.microsoft.com/en-us/projects/joulemeter/default.aspx

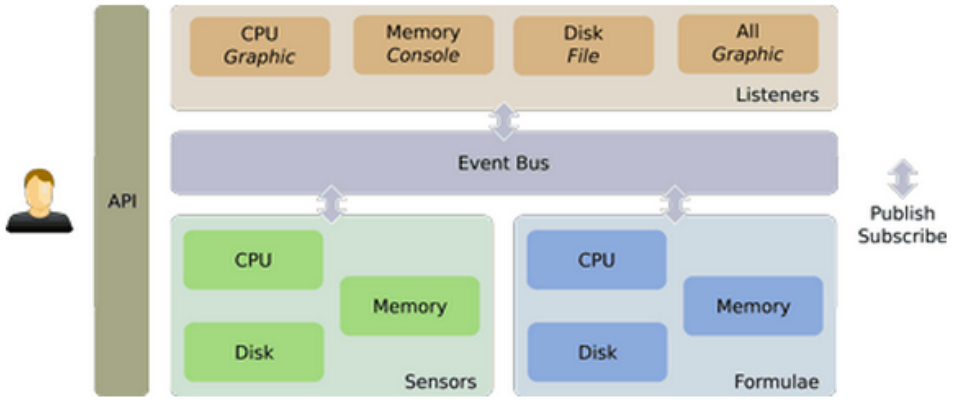

FIGURE 2.5 – Architecture de PowerAPI (Figure extraite de [Bourdon et al., 2013])

La figure 2.6 liste les différents paramètres utilisés dans leurs modèles. Par exemple, le temps pour sauvegarder un point de reprise est considéré comme étant toujours égal à 180 secondes alors que le temps pour le restaurer est de 30 secondes. Basées sur des approximations théoriques des paramètres, ces estimations ne sont donc pas du tout liées à l'application exécutée.

Par exemple, pour l'estimation énergétique de la sauvegarde et la récupération de points de reprise, le modèle considéré est :

$$E = WSH + (\frac{W}{\tau} - 1) \cdot \delta SL + \frac{T}{M}(\delta SL + \frac{\tau - \delta}{2} SH) + \frac{T}{M} RSL$$

Les différents paramètres de ce modèle sont détaillés dans la figure 2.6. Dans ce modèle, ils considèrent donc que la puissance électrique est égale soit à SH, où H vaut 100 W soit à SL, où L vaut 40 W (voir figure 2.6).

Bien que ces outils et techniques peuvent fournir des informations assez intéressantes par exemple en repérant les tâches les plus gourmandes en énergie, **elles ne fournissent pas des estimations assez précises** dans la mesure où elles ne sont pas suffisamment corrélées aux applications et qu'elles ne tiennent de tous les paramètres qui peuvent rentrer en jeu dans l'évaluation de la consommation énergétique. Par ailleurs, **elles ne répondent pas à notre besoin qui est d'obtenir une estimation énergétique d'un service applicatif avant de l'exécuter**.

2.2 Solutions pour une consommation efficace de l'énergie

La question de l'efficacité énergétique dans les plates-formes distribuées a été explorée surtout dans le contexte des réseaux, centres de données, ou l'informatique dans le nuage. Quelles sont les solutions mises en oeuvre pour consommer moins et mieux dans une plate-forme de calcul haute performance ?

Parameter	Description	Value
V	Optimal virtualization ratio	> 8
W	Time to solution with V	25 h
M	Mean-time-to-interrupt of the system	-
S	Total number of sockets in the system	-
δ	Checkpoint time	180 s
τ	Optimum checkpoint period	-
R	Restart time	30 s
T	Total execution time	-
E	Total energy consumption	-
μ	Message-logging slowdown	1.02
P	Available parallelism during recovery	8
ϕ	Message-logging recovery speedup	1.2
σ	Parallel recovery speedup	P
λ	Parallel recovery slowdown	$\frac{P+1}{P}$
H	Max power of each socket	100 W
L	Base power of each socket	40 W

TABLE II
PARAMETERS OF ENERGY MODEL

FIGURE 2.6 – Paramètres du modèle énergétique considéré par [Meneses et al., 2012] (Figure extraite de [Meneses et al., 2012])

2.2.1 Solutions pour réduire la consommation énergétique

Différentes approches ont été proposées afin de réduire la consommation énergétique. Certaines ont cherché à toucher la sensibilité des utilisateurs afin qu'ils prennent la décision de réduire leur consommation énergétique. D'autres ont proposé d'adapter la consommation énergétique des machines en fonction de leurs usages.

2.2.1.1 Sensibiliser les utilisateurs à faire de bons choix

Une des approches pour consommer moins d'énergie dans le contexte du calcul très haute performance est de chercher à sensibiliser l'utilisateur. Pour cela, l'approche consiste à fournir à l'utilisateur des informations sur la consommation énergétique des applications qu'il exécute, ce qui peut l'amener à réduire lui-même son utilisation des ressources.

Dans [Da-Costa et al., 2010], l'architecture GREEN-NET [12] (voir figure 2.7) interagit avec les utilisateurs afin de réduire la consommation énergétique tout en respectant leurs exigences. Pour cela, elle utilise des wattmètres externes pour mesurer et fournir aux utilisateurs en temps réel les mesures de consommation énergétique des noeuds de calcul, et ce dans l'optique de toucher leurs sensibilités.

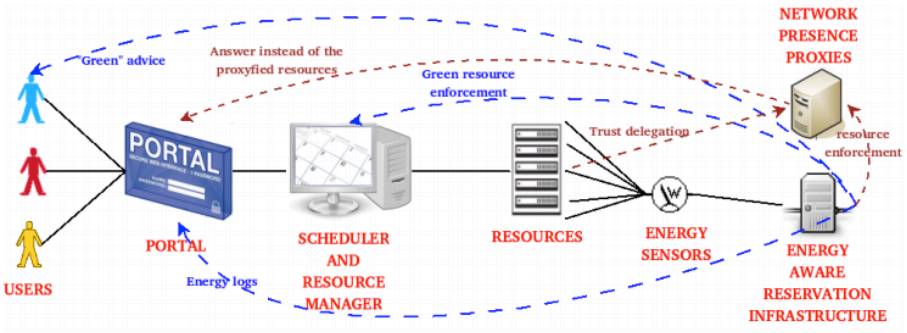

FIGURE 2.7 – Architecture GREEN-NET (Figure extraite de [Da-Costa et al., 2010])

Les utilisateurs consultent sur une page web les mesures énergétiques en temps réel en temps réel. De plus, GREEN-NET invite les utilisateurs à prendre des décisions visant à réduire la consommation énergétique. En effet, une fois qu'ils sont informés de leur consommation énergétique, les utilisateurs peuvent agir :

- soit en indiquant explicitement s'il est possible de relâcher quelques serveurs de calcul ;
- soit en demandant à GREEN-NET d'agir plus ou moins fortement en termes d'économies d'énergie en acceptant une des six politiques de réduction de la consommation. Les six politiques de réduction de la consommation consistent à "coller" les réservations de ressources afin de minimiser les creux dans l'agenda

12. Projet GREEN-NET : http://www.ens-lyon.fr/LIP/RESO/Projects/green-net

de réservation et ainsi favoriser l'application des approches d'extinction et rallumage avec une durée suffisamment importante (voir section 2.2.1.2).

Pour consommer moins, il faut donc que l'utilisateur sacrifie des ressources de calcul ou/et accepte de déplacer ses réservations. L'énergie est donc économisée par GREEN-NET aux dépends de la qualité de service offerte aux utilisateurs. Dans le chapitre 5, en utilisant le même nombre de noeuds, nous montrons qu'il est possible de consommer moins en choisissant les noeuds d'une grappe homogène qui consomment le moins d'énergie. De plus, l'approche que nous présentons dans le chapitre 6 permet de consommer moins et mieux l'énergie tout en veillant à ce que le retardement de la réservation de ressources soit tolérable par l'utilisateur.

Par ailleurs, le défi que nous cherchons à relever est de permettre aux utilisateurs de choisir parmi les différentes versions d'un service applicatif, celle qui consomme le moins d'énergie sans que cela n'affecte les performances et la fiabilité de l'exécution de son application.

Les travaux précédents ne se sont pas intéressés à cette question d'un point de vue énergétique. Ils ont par contre cherché à optimiser les performances du service applicatif. Par exemple, dans les protocoles de tolérance aux pannes, certains travaux comme [Young, 1974, Daly, 2006] se sont intéressés à établir un modèle théorique permettant de fournir une approximation de l'intervalle optimal entre deux points de reprise successifs. Cet intervalle est défini en minimisant le coût lié à la sauvegarde de points de reprise tout en assurant la fiabilité optimale.

La première approximation théorique de cet intervalle a été proposée par Young [Young, 1974] :

$$\tau_{opt} = \sqrt{2\delta M} \tag{2.1}$$

δ est le temps nécessaire pour sauvegarder un point de reprise. M est le temps moyen avant la prochaine défaillance du système. Il correspond au MTTI [13].

Dans [Sloot et al., 2003, Daly, 2006] Daly a proposé deux nouvelles approximations pour estimer au mieux la valeur de l'intervalle optimal entre deux points de reprise successifs. Selon [Sloot et al., 2003], cet intervalle est égal à $\sqrt{2\delta(M+R)} - \delta$, où R est le temps nécessaire pour restaurer un point de reprise.

Selon [Daly, 2006], une meilleure approximation de cet intervalle est :

$\tau_{opt} = \sqrt{2\delta M}[1 + \frac{1}{3} \cdot (\frac{\delta}{2M})^{1/2} + \frac{1}{9}(\frac{\delta}{2M})] - \delta$; pour $\delta < 2M$

$\tau_{opt} = M$; pour $\delta >= 2M$

Bien que ces modèles fournissent des approximations théoriques de plus en plus réalistes, l'intervalle optimal dépend fortement du matériel utilisé dans le système. De plus, ces intervalles optimaux ne prennent pas en considération le critère de la consommation énergétique.

13. MTTI : Mean Time To Interrupt

2.2.1.2 Adapter la consommation énergétique des serveurs en fonction de leur usage

En concevant des composants efficaces en énergie, les systèmes informatiques peuvent eux-mêmes consommer moins d'énergie. [Barroso and Hölzle, 2007] revendique que la puissance électrique dissipée par un serveur doit être proportionnelle à son usage, c'est-à-dire que le serveur ne doit (quasiment) rien consommer quand il n'exécute rien et, éventuellement consommer plus dans le cas où il exécute des programmes. Ainsi, avec des serveurs dont l'énergie consommée est proportionnelle à l'usage, un centre de données pourrait économiser jusqu'à 50 % d'énergie.

Combinées à des optimisations matérielles imaginées par les constructeurs, les approches visant à réduire la consommation d'énergie des plates-formes distribuées peuvent être organisées en deux classes distinctes :

– l'approche par extinction/rallumage
– l'approche par ralentissement

Dans une plate-forme de calcul haute performance, les serveurs consomment quasiment autant d'énergie lorsqu'ils sont au repos ou lorsqu'ils exécutent des applications. Pendant les périodes de basse utilisation, certains serveurs pourraient être éteints ou seulement mis dans des états de très faible consommation énergétique. Ainsi, l'approche par extinction/rallumage consiste à éteindre dynamiquement des ressources inutilisées et à les rallumer seulement quand elles sont requises. Cette stratégie consiste donc à minimiser le nombre de serveurs allumés tout en respectant la qualité de service requise par l'utilisateur.

De nombreux travaux comme [Chase et al., 2001, Fan et al., 2007a, Hikita et al., 2008], sont basés sur cette approche et suggèrent l'utilisation d'algorithmes d'extinction et de rallumage afin d'éviter que les machines ne consomment de l'énergie alors qu'elles sont au repos. Dans [Hikita et al., 2008], les auteurs se sont basés sur cette approche pour présenter une étude expérimentale d'une durée de 45 mois. Ils ont utilisé un algorithme assez simple : si un noeud est resté au repos pendant 30 minutes alors il est mis sous tension. Lorsque ce noeud est requis pour exécuter une application, il est allumé et mis en activité. Dans le cas de leur système, le rallumage d'un noeud est très long dans la mesure où il dure approximativement 45 minutes à cause d'un diagnostic matériel ayant lieu à chaque redémarrage. Toutefois, la stratégie décrite peut permettre une réduction de la consommation énergétique de 39 % au meilleur des cas.

Pinheiro et al. [Pinheiro et al., 2001] ont aussi développé une approche qui éteint dynamiquement des noeuds et les rallume. Cette approche utilise la concentration de la charge de travail sur un minimum de noeuds afin que les noeuds inutilisés soient mis sous tension. Leurs tests expérimentaux ont été réalisés sur une grappe de 8 noeuds et ont révélé qu'il était possible d'économiser 19 % d'énergie. Dans leur configuration, ils ont considéré qu'il fallait 100 secondes pour allumer un serveur et 45 secondes pour l'éteindre. Leur approche tolère une dégradation de performances (allongement du temps d'exécution) de 20 %. Cependant, toutes les expériences considérées ne démarrent qu'avec un seul noeud allumé, ce qui suppose que tous les autres noeuds sont déjà éteints. Les noeuds ne sont allumés que lorsqu'ils sont

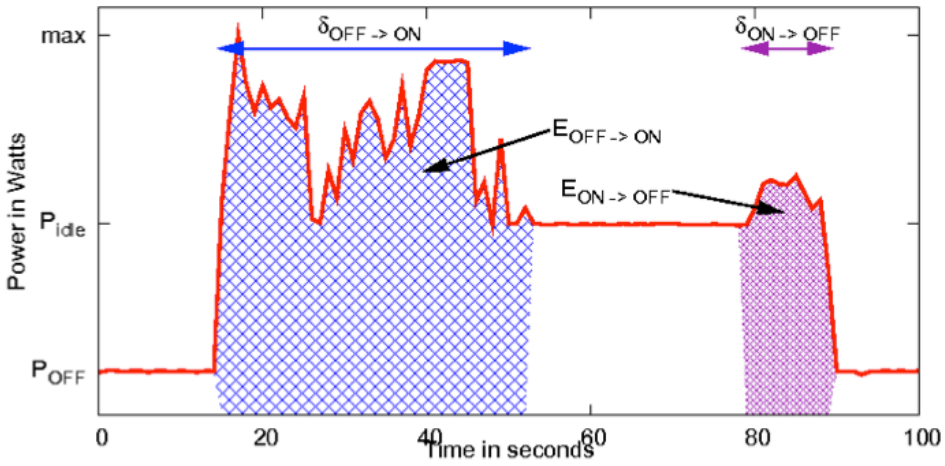

FIGURE 2.8 – Prise en compte de l'énergie d'extinction et de rallumage (Figure extraite de [Orgerie et al., 2008b])

requis.

Afin de mettre en oeuvre les algorithmes d'extinction et de rallumage, plusieurs travaux ont essayé de pratiquer la concentration de charge en migrant les tâches d'un noeud à un autre [Orgerie et al., 2008b] ou en migrant des machines virtuelles sur d'autres noeuds physiques [Hermenier et al., 2006, Orgerie and Lefèvre, 2009]. Toutefois, ces migrations sont souvent complexes à réaliser dans le mesure où il faut s'assurer que les tâches à migrer soient divisibles (pour les migrer en pleine exécution) et que le noeud vers lequel la tâche sera migrée soit synchronisé avec le noeud à partir duquel on cherche à migrer la tâche.

De plus, ces migrations génèrent un surcoût en termes d'énergie et de performance, qu'il est essentiel à prendre en compte lors du calcul d'efficacité énergétique. Dans [Orgerie et al., 2008b], Orgerie et al. proposent de prendre en compte dans leur approche les pics de consommation qui ont lieu lors du redémarrage des noeuds éteints. À cette fin, les auteurs présentent un modèle qui prédit l'utilisation future des ressources et définit une durée minimale à partir de laquelle une ressource consomme moins d'énergie si elle est éteinte pendant cette durée. La figure 2.8 met en évidence la nécessité de tenir compte de cette durée minimale.

Afin que l'on économise de l'énergie avec l'approche d'extinction et de rallumage, il est indispensable que l'énergie qui aurait due être consommée si la machine était allumée (au repos) soit supérieure à l'énergie consommée à cause de l'extinction et du rallumage. Ainsi, la durée minimale établie par [Orgerie et al., 2008b] est telle que :

$$T_s = \frac{E_s - P_{OFF} \cdot (\delta_{ON->OFF} + \delta_{OFF->ON}) + E_{ON->OFF} + E_{OFF->ON}}{P_{repos} - P_{OFF}} \quad (2.2)$$

E_s est l'énergie minimum que l'on souhaite économiser. Par exemple si E_s est

égale à 10 joules, cela veut dire que l'on considère que ça ne vaut pas la peine d'éteindre et de rallumer si on n'économise que 10 joules ! $\delta_{ON->OFF} + \delta_{OFF->ON}$ est la durée totale pour éteindre puis rallumer le serveur à éteindre. $E_{ON->OFF} + E_{OFF->ON}$ est la consommation énergétique du serveur pendant cette durée. P_{OFF} est sa puissance électrique lorsqu'il est éteint alors que P_{repos} est sa consommation au repos (lorsque rien n'est exécuté sur le serveur à l'exception du système d'exploitation).

S'il est prévu qu'une ressource soit inactive pendant une période supérieure à cette durée minimum, alors elle doit être éteinte pendant cette période. Sinon, elle doit rester allumée. L'approche par extinction/rallumage donne des résultats satisfaisants en termes d'économies d'énergie. Cependant, sa mise en oeuvre nécessite la définition d'un modèle de prévision suffisamment fiable et implique la gestion de la perte de connectivité avec les ressources éteintes.

L'approche de ralentissement consiste à ajuster dynamiquement le niveau de performance d'une ressource en fonction des performances requises par l'application. Étant donné que la puissance électrique du processeur représente une part prépondérante de la consommation totale du système (environ 50 % [Ge et al., 2009]) lorsque le système est actif, de nombreuses études sont basées sur cette approche en proposant d'utiliser les techniques DVFS (Dynamic Voltage and Frequency Scaling) pour les processeurs [Hotta et al., 2006, Etinski et al., 2010, Hsu and chun Feng, 2005, Freeh et al., 2007] ou la technique ALR (Adaptive Link Rate) pour les interfaces réseaux [Gunaratne and Christensen, 2006]. Dans l'approche ALR, il est proposé d'adapter la performance du réseau en fonction des besoins en bande passante. Dans les approches DVFS, il est proposé d'adapter le plus précisément possible la vitesse d'horloge du processeur en fonction des performances requises par l'application. En utilisant un processeur à faibles fréquences et voltages, la puissance électrique utilisée peut être réduite, mais le temps d'exécution de l'application peut-être augmenté.

Les techniques DVFS ont inspiré la définition des différents états d'énergie caractérisés par la fréquence du processeur, le voltage et la consommation d'énergie. Ces techniques sont de plus en plus prises en compte dans la mise en oeuvre de processeurs aujourd'hui surtout avec l'ACPI [14] qui définit des états de consommation pour le processeurs appelés *C-states* allant de C_0 jusqu'à C_n où C_0 est l'état d'intense activité (et donc de consommation énergétique élevée) et C_n est l'état de plus basse consommation énergétique. Des états énergétiques équivalents sont proposés pour les autres périphériques et sont appelés *D-states*.

Étant donné que l'approche extinction/rallumage et l'approche ralentissement sont très dépendantes de l'usage, le défi est de savoir quand est-ce qu'il faut ralentir, éteindre et rallumer. Pour cela, il est indispensable d'associer à ces approches des modèles de prédiction qui permettent de repérer les périodes d'inactivité. Pour être capable d'établir des modèles de prédiction suffisamment fiables, il est important de

14. ACPI : Advanced Configuration and Power Interface : http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf

comprendre comment évolue la consommation énergétique des serveurs de calcul.

De plus, pour appliquer ces deux approches, il est nécessaire que ces périodes d'inactivité soient suffisamment longues. Ceci va à l'encontre du calcul très haute performance où l'on cherche à aller au plus vite en utilisant au maximum les capacités des ressources de calcul.

Les travaux précédents ne nous permettent pas de savoir quels sont les noeuds de calcul à éteindre ou à ralentir si on a le choix parmi plusieurs noeuds. Nous verrons dans le chapitre 5 que le choix des ressources à éteindre peut avoir un impact important sur la quantité d'énergie inutilisée (et donc qui peut être économisée).

2.2.2 Solutions pour mieux consommer l'énergie

Pour "consommer mieux", il est nécessaire d'interagir avec le fournisseur d'énergie afin de savoir quand est ce que l'énergie est la moins chère et la moins polluante. Comment maintenir ce dialogue avec le fournisseur d'énergie ? Comment planifier les exécutions pour tenir compte également du coût financier et de l'impact environnemental ?

2.2.2.1 Des grilles intelligentes de distribution électrique au service d'une meilleure coordination entre le fournisseur d'énergie et ses consommateurs

Actuellement, le réseau de distribution électrique ne met pas en place un réel dialogue entre le fournisseur et les consommateurs. D'une part, les informations sur la consommation énergétique des utilisateurs ne remontent pas au fournisseur d'énergie. D'autre part, les informations sur la production de l'électricité (par exemple, les proportions des différentes sources d'énergie) ne sont pas communiquées en temps réel aux consommateurs. Par conséquent, la production et la distribution de l'électricité ne sont pas adaptées de façon optimale en fonction des besoins des utilisateurs.

Les grilles intelligentes de distribution d'électricité ou "smart grids" [European Commission, 2006] font référence à l'ensemble des solutions informatiques mises en oeuvre pour optimiser les interactions entre le fournisseur d'énergie et ses clients pour une meilleure consommation énergétique. L'introduction de smart grids nécessite la mise en place d'une infrastructure de compteurs électriques intelligents permettant la remontée au fournisseur d'énergie d'informations liées à la consommation énergétique des utilisateurs [Backer, 2007].

Ainsi, grâce aux prévisions de la consommation énergétique de tous ses clients, le fournisseur d'énergie serait capable d'adapter la production à la consommation et l'offre tarifaire en fonction de la demande. Plusieurs travaux ont mis en évidence les intérêts d'une meilleure coordination entre l'offre et la demande en électricité : production d'énergies moins polluantes [Fu et al., 2012, Saber and Venayagamoorthy, 2012], tarification de l'énergie plus pragmatique [Thimmapuram and Kim, 2013, Yu et al., 2012], réduction des pannes électriques [Chertkov et al., 2011, Calderaro et al., 2011] et diminution des pics de consommation [Fan et al., 2007b].

Toutefois, la mise en place d'une telle infrastructure nécessite de garantir :

– La fiabilité et la robustesse du réseau [Gungor et al., 2013]. Il faut assurer

que des quantités colossales de données soient transmises en temps réel sur le réseau (venant du fournisseur d'énergie et des différents clients).
- La sécurité et la confidentialité des données acheminées [Cleveland, 2008, Lisovich et al., 2010]. D'une part, les données transmises sur le réseau peuvent être sujettes à des attaques de pirates souhaitant les déformer ; par exemple, en cherchant à réduire les consommations énergétiques transmises au fournisseur d'énergie pour diminuer sa facture d'électricité personnelle. D'autre part, les données véhiculées par le réseau sont suffisantes pour avoir un portrait détaillé de l'activité d'un foyer : les moments où les personnes se réveillent, mangent, prennent leurs douches et s'endorment [Lisovich et al., 2010].
- Le stockage et le traitement de ces données [Navarro et al., 2011, Xu et al., 2012]. Une des approches pour optimiser le stockage est de compresser les données. Cependant, elle peut entraîner une perte d'information et un coût supplémentaire en temps et en énergie.

La plupart des travaux se sont intéressés à étudier la mise en place d'une smart grid dans le contexte des équipements électriques utilisés dans les habitations individuelles [Ozturk et al., 2013, Tsui and Chan, 2012, Ali et al., 2012] et des stations de service pour les véhicules électriques [Saber and Venayagamoorthy, 2012, Ko and Hahn, 2013]. Les applications considérées par la smart grid visent donc une multitude de petits consommateurs. Dans notre cas, nous considérons la mise en place d'une smart grid dans le contexte d'une infrastructure de calcul très haute performance qui nécessite donc une puissance électrique très élevée (de l'ordre du mégawatt, soit environ la puissance électrique consommée par un village de 7300 d'habitants).

Dans cette thèse, nous ne cherchons pas à proposer des solutions aux différents défis mais nous nous appuyons sur une telle infrastructure pour mettre en évidence comment la coordination entre le fournisseur d'énergie et les gros consommateurs favorise une meilleure consommation énergétique. Dans notre cas, les gros consommateurs sont les plates-formes de calcul distribuées à très larges échelles.

2.2.2.2 Ordonnancement des réservations de ressources pour une consommation énergétique meilleure

De nombreux algorithmes d'ordonnancement vert des tâches ou des ressources ont été proposés dans les travaux précédents. Dans un algorithme d'ordonnancement vert, les critères mis en avant sont la réduction de la consommation énergétique [Pinheiro et al., 2001], la minimisation des coûts financiers [Kim and Poor, 2011, Zeng et al., 2012] ou la réduction de la pollution de l'environnement [Goiri et al., 2011, Liu et al., 2012].

Un des premiers travaux sur l'ordonnancement efficace en énergie a été proposé dans [Pinheiro et al., 2001] . Dans cet article, Pinheiro et al. ont proposé une technique pour la gestion d'une grappe de machines physiques avec l'objectif de minimiser la consommation d'énergie, tout en assurant la qualité de service requise. La principale technique pour minimiser la consommation d'énergie est d'ordonnancer les réservations des utilisateurs sur un minimum de noeuds physiques afin d'éteindre

le maximum de ressources de calcul.

Les auteurs du papier [Aupy et al., 2012] se sont intéressés à minimiser la consommation énergétique tout en respectant des contraintes de délais et de résilience. Pour réduire la consommation énergétique, ils suggèrent d'appliquer la technique DVFS. Dans leur étude, ils considèrent que même si DVFS permet de réduire la consommation de l'énergie, cette technique augmente le temps d'exécution et peut dégrader les processeurs. La dégradation des processeurs se traduit par une accroissement de la fréquence des défaillances, qui peut entraîner une augmentation des ré-exécutions d'applications et donc une hausse de la consommation énergétique. Ils suggèrent ainsi que les approches visant à ordonnancer des tâches dans l'optique d'économiser de l'énergie, doivent prendre en compte ces deux contraintes (augmentation du temps d'exécution et accroissement des défaillances) résultant de la dégradation du matériel. Il s'agit de contraintes dans la mesure où ils peuvent avoir l'effet inverse : augmentation de la consommation énergétique.

Dans [Kim and Poor, 2011], les auteurs considèrent que le prix de l'énergie varie dans le temps. L'ordonnanceur qu'ils considèrent, a accès aux tarifs actuels et passés, et a une connaissance statistique sur les prix futurs. Les tâches considérées ont une durée plus ou moins longue et peuvent être soit divisibles soit indivisibles. Les auteurs suggèrent de réduire le coût financier du point de vue des utilisateurs, en planifiant leurs tâches aux moments où l'énergie est moins chère. Dans leurs simulations, ils ont considéré les prix réels du fournisseur d'électricité Commonwealth Edison [15]. Ils montrent que les utilisateurs peuvent ainsi faire d'importantes économies et spécialement lorqu'ils exécutent des tâches courtes.

Dans un autre exemple [Goiri et al., 2011], les auteurs proposent GreenSlot, un ordonnanceur vert des tâches d'un centre de données. Le centre de données considéré est alimenté par un panneau solaire photovoltaïque et par le réseau de distribution de l'électricité. GreenSlot prédit la quantité d'électricité produite par le panneau solaire disponible dans un futur proche et planifie les tâches avec l'objectif de maximiser la consommation de l'énergie peu polluante tout en respectant les délais fixés par les utilisateurs. Leurs résultats montrent que GreenSlot permet d'optimiser la consommation de l'énergie solaire qui est disponible.

Bien qu'ils cherchent à ordonnancer les tâches ou les réservations de ressources de calcul afin de "consommer mieux", ces travaux antérieurs ne proposent pas d'optimiser simultanément la consommation d'énergie, le coût financier et l'impact de la pollution.

2.3 Conclusions du chapitre

Tout d'abord, nous avons étudié les différents travaux cherchant à mesurer la puissance électrique lors de l'exécution des applications. Pour cela, nous avons iden-

15. Commonwealth Edison est le plus grand fournisseur d'électricité de l'État d'Illinois aux États-Unis

tifié et décrit les différents dispositifs utilisés. De plus, nous avons mis en évidence les approches mises en oeuvre pour collecter les mesures de puissance électrique à l'aide de ces dispositifs. Ces travaux de recherche ne répondent pas à la question de savoir s'il convient mieux d'utiliser un dispositif de mesure plutôt qu'un autre en fonction de l'usage que l'on souhaite en faire. Dans le chapitre 3, nous nous intéresserons à comparer quelques équipements de mesure pour savoir lesquels sont les plus appropriés pour mesurer la puissance électrique des services applicatifs. Par ailleurs, les travaux présentés ne se sont pas intéressés à mesurer la consommation énergétique des services applicatifs tels que les protocoles de tolérance aux pannes ou les algorithmes de diffusion de données. Nous présenterons dans le chapitre 3 notre démarche pour mesurer la consommation énergétique pour un service applicatif donné.

Comme évoqué dans le chapitre 1, nous avons besoin d'estimer la consommation de ces services applicatifs avec précision et sans pré-exécuter l'application. Les travaux qui ont proposé des approches d'estimation ne répondent pas à notre besoin. D'une part, certaines approches considèrent que la consommation énergétique est uniquement corrélée aux performances et s'appuient sur les compteurs de performance fournies par le matériel utilisé. Ces approches ne fournissent pas des estimations assez précises dans la mesure où elles ne tiennent pas compte de la variabilité des différents composants matériels et qu'elles sont intrusives : elles nécessitent l'exécution de programmes collectant les valeurs de compteurs. L'autre inconvénient est que les compteurs de performance utilisés pour l'estimation sont très souvent dépendant de la marque des processeurs. Les modèles utilisés ne sont donc pas génériques et doivent être adaptés à chaque nouvelle plate-forme. D'autre part, les autres approches s'appuient soit sur les différents compteurs de performance logiciels soit sur le taux d'utilisation des différents composants matériels. Ces approches ne répondent pas à notre besoin qui est d'obtenir une estimation énergétique d'un service applicatif avant d'exécuter l'application. Dans le chapitre 4, le défi que nous cherchons à relever est d'estimer précisément la consommation énergétique de ces services applicatifs sans pré-exécuter l'application, et ce en se basant sur des mesures réelles de la puissance électrique et des temps d'exécution liés aux services applicatifs.

Ensuite, nous avons étudié les différentes approches visant à réduire la consommation énergétique dans des infrastructures de calcul très haute performance. Nous avons mis en évidence deux familles d'approches : d'une part, celles qui cherchent à sensibiliser l'utilisateur à prendre des décisions dans l'optique de réduire sa consommation énergétique ; et d'autre part, celles qui appliquent des leviers énergétiques (extinction et ralentissement) en fonction de l'usage des machines. Pour réduire la consommation énergétique, les travaux basés sur la première famille d'approches poussent les utilisateurs à faire des choix aux dépends de la qualité de service : choisir moins de noeuds pour consommer moins et/ou retarder les réservations de ressources de plusieurs heures pour créer des conditions favorables à l'application de l'approche extinction/rallumage (éteindre le plus longtemps et le moins souvent). Dans les chapitres 5 et 6, le défi que nous cherchons à relever est de permettre aux

utilisateurs de faire des choix sans pour autant dégrader cette qualité de service, c'est-à-dire sans réduire le nombre de noeuds demandés, avec un retardement des réservations qui reste tolérable par l'utilisateur et sans dégrader le service applicatif utilisé (par exemple, sans réduire la fiabilité des protocoles de tolérance aux pannes). Quant aux approches basées sur l'extinction ou le ralentissement des ressources, les travaux présentés **ne nous permettent pas de savoir lesquelles éteindre ou ralentir si on a le choix parmi plusieurs ressources**. Par ailleurs, l'extinction de ressources de calcul est difficilement applicable pendant un calcul très haute performance étant donné que les périodes d'inactivité ne sont souvent pas assez longues. Dans le cas où elles sont suffisamment longues (supérieures au seuil minimal suggéré par [Orgerie et al., 2008b]), on pourrait économiser de l'énergie grâce à l'extinction des ressources inutilisées. Dans le cas contraire, nous pourrons toujours les ralentir en appliquant les techniques DVFS pendant ces périodes d'inactivité.

En plus des solutions visant à réduire la consommation énergétique, nous nous sommes intéressés à étudier les travaux qui cherchent à minimiser les coûts financiers et la pollution de l'environnement. Pour cela, l'utilisateur a besoin de savoir quand est-ce que l'énergie est moins chère et/ou moins polluante. À cette fin, une coordination entre le fournisseur d'énergie et les utilisateurs est indispensable. Cette coordination est assurée grâce à la mise en place d'une grille intelligente de distribution électrique. Les travaux présentés soulèvent plusieurs défis à relever pour mettre en place une telle infrastructure. Notre but n'est pas de proposer des solutions à ces différents défis mais nous nous appuierons sur une telle infrastructure pour montrer dans le chapitre 5, comment une meilleure consommation énergétique peut être favorisée grâce la coordination entre le fournisseur d'énergie et les plates-formes de calcul distribuées à très larges échelles. Par ailleurs, d'autres travaux ont proposé des algorithmes d'ordonnancement vert qui visent la minimisation de la consommation énergétique, des coûts financiers ou de la pollution de l'environnement. Cependant, ces travaux ne proposent pas d'optimiser simultanément ces trois critères.

3 Mesure de la puissance électrique de services applicatifs

Afin de comprendre et de prédire le comportement électrique des services applicatifs, il est important d'être capable de mesurer leur puissance électrique. Dans un premier temps, nous montrons dans la section 3.1 comment nous procédons pour mesurer la puissance électrique dans les deux services applicatifs auxquels nous nous intéressons, à savoir la tolérance aux pannes [Diouri et al., 2012b] et la diffusion de données [Diouri et al., 2013g].

La section 3.1 nous amène à nous poser plusieurs questions :

1. Quel dispositif de mesure doit-on utiliser pour obtenir des mesures suffisamment fiables de la puissance électrique ? Quelle est la fréquence de mesure dont on a besoin pour comprendre le comportement électrique des applications ? [Diouri et al., 2013a]

2. A-t-on besoin de mesurer la puissance électrique de toutes les machines d'une même grappe (identiques d'un point de vue matériel) ? Ont-elles la même puissance électrique si elles exécutent les mêmes tâches ? [Diouri et al., 2013d]

Afin de répondre à la première interrogation, la section 3.2 présente une comparaison de différents équipements de mesure de puissance électrique. Quant à la section 3.3, elle permet de répondre à la deuxième interrogation grâce à la confrontation des mesures électriques que nous obtenons pour plusieurs machines identiques issues d'une même grappe de calcul.

3.1 Mesure de la puissance électrique dans les protocoles de tolérance aux pannes et dans les algorithmes de diffusion de données

Dans cette section, nous cherchons à mesurer la puissance électrique de certaines opérations de base que nous retrouvons dans les services relatifs à tolérance aux pannes et dans la diffusion de données. La section 3.1.1 présente le dispositif expérimental que nous utilisons. La section 3.1.2 présente les mesures électriques que nous obtenons pour les opérations de base que nous retrouvons dans la tolérance aux pannes. La section 3.1.3 présente les mesures de la puissance électrique que nous obtenons dans le cas de la diffusion de données. La section 3.1.4 compare les mesures de la puissance électrique que nous avons relevées à celles que nous obtenons pour différents bancs d'essai sollicitant les mêmes ressources. Cette section synthétise les observations et conclusions que nous tirons de nos mesures électriques.

3.1.1 Dispositif expérimental et méthodologie

Pour mesurer la puissance électrique des protocoles de tolérance aux pannes et des algorithmes de diffusion de données, nous avons utilisé différentes grappes de calcul de la plate-forme expérimentale à large échelle Grid'5000 [1] [Cappello et al., 2005]. La figure 3.1 représente une vue d'ensemble de la grille Grid'5000. Cette grille interconnecte 11 sites répartis sur toute la France et le Luxembourg. Chaque site possède une ou plusieurs grappes interconnectées par différents types de réseaux. Les caractéristiques de chacune des trois grappes considérées sont détaillées dans le tableau 3.1.

Nous supervisons la puissance électrique des noeuds des grappes *Sagittaire* et *Taurus* (localisées à Lyon) à l'aide d'une infrastructure de mesure de la puissance électrique, qui est constituée de plusieurs wattmètres externes fabriqués par la société Omegawatt [2]. Cette infrastructure de mesure électrique est décrite dans [Dias de

1. La majeure partie des expériences présentées dans cette thèse ont été réalisées en utilisant la plateforme expérimentale Grid'5000, issue de l'Action de Développement Technologique (ADT) Aladdin pour l'INRIA, avec le support du CNRS, de RENATER, de plusieurs Universités et autres contributeurs (https://www.grid5000.fr)

2. Omegawatt : www.omegawatt.fr

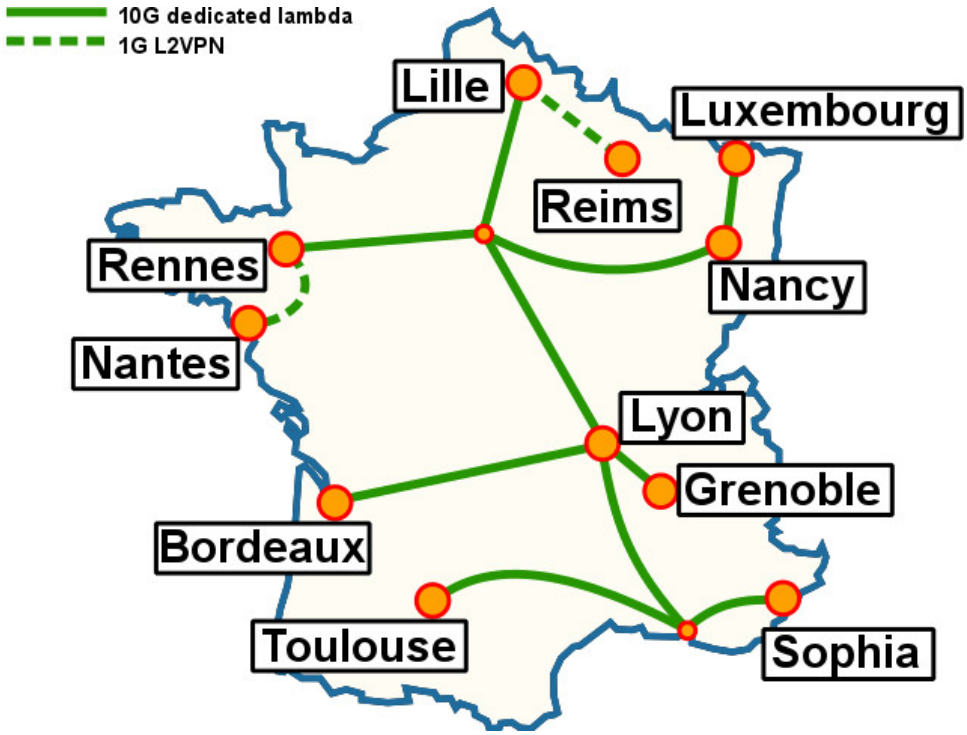

FIGURE 3.1 – Vue d'ensemble de la grille Grid'5000

Assuncao et al., 2010b] ; elle permet d'obtenir pour chaque noeud supervisé, la puissance électrique moyenne en watts calculée chaque seconde à partir d'environ 6000 échantillons de puissance électrique instantanée mesurés en une seconde [Dias de Assuncao et al., 2010a].

Les enregistrements effectués par cette infrastructure sont affichés en temps réel et sont sauvegardés dans une base de données, dans l'optique de permettre aux utilisateurs d'obtenir la puissance électrique et l'énergie consommée par une ou plusieurs machines entre une date de début et une date de fin.

Nous supervisons la puissance électrique des noeuds de la grappe *Stremi* (localisée à Reims) à l'aide d'une infrastructure d'unités de distribution d'électricité (PDU) Raritan. Ces PDUs fournissent une mesure de la puissance électrique en watts chaque trois secondes pour chaque noeud supervisé. Les enregistrements de puissance électrique provenant de ces PDUs sont sauvegardés dans une base de données et peuvent être obtenues par le biais de requêtes SNMP [3].

3. SNMP : Simple Network Management Protocol ou *protocole simple de gestion du réseau*

Nom de la grappe	***Sagittaire***	***Stremi***	***Taurus***
Localisation	Lyon	Reims	Lyon
Année d'installation	début 2005	début 2011	fin 2012
Système d'exploitation	Debian 6 (Squeeze)	Debian 6 (Squeeze)	Debian 6 (Squeeze)
Nombre de noeuds identiques	60 Sun Fire V20z	42 HP Proliant DL165 G7	16 Dell R720
Nombre de CPUs par noeud	2 AMD Opteron 250 2.4 GHz	2 AMD Opteron 6164 1.7GHz	2 Intel Xeon 2.3 GHz
Nombre de coeurs par noeud	2 coeurs	24 coeurs	12 coeurs
Mémoire	2 Go	48 Go	32 Go
Disque dur	73 Go SCSI	250 Go SATA	598 Go SCSI
Réseau	Gigabit Ethernet	Gigabit Ethernet	10 Gigabit Ethernet
Wattmètres utilisés	OmegaWatt	PDU Raritan	OmegaWatt
Précision des mesures	+/- 0.125 W	+/- 6 W	+/- 0.125 W
Fréquence de mesure	1 mesure par seconde	1 mesure toutes les 3 secondes	1 mesure par seconde
Type de mesure	puissance moyenne	puissance moyenne	puissance moyenne

TABLE 3.1 – Caractéristiques techniques des trois grappes de calcul utilisées dans nos expérimentations

Étant donné que les wattmètres fournissent une mesure par seconde pour les grappes *Sagittaire* et *Taurus* et une mesure toutes les trois secondes pour la grappe *Stremi*, nous avons besoin que le temps d'exécution d'une opération de base du service applicatif dont nous souhaitons mesurer la puissance électrique, soit au moins supérieur à une seconde pour les grappes *Sagittaire* et *Taurus* et au moins supérieur à trois secondes pour la grappe *Stremi*.

Pour avoir plusieurs points de mesure, il est nécessaire que ce temps d'exécution soit de quelques secondes. Pour ce faire, nous essayons dans un premier temps de jouer sur les paramètres d'exécution afin que l'opération pour laquelle nous cherchons à mesurer la puissance électrique dure quelques secondes. Si cela n'est pas possible, nous instrumentons l'implémentation du service applicatif afin de répéter spécifiquement l'opération pour laquelle nous voulons mesurer la puissance électrique.

3.1.2 Puissance électrique dans les protocoles de tolérance aux pannes

Pour les mesures électriques, nous nous sommes focalisés sur deux catégories de protocoles de tolérance aux pannes : les protocoles coordonnés et les protocoles non coordonnés. Dans toutes nos mesures électriques liées à ces deux protocoles, nous n'avons utilisé qu'un seul coeur par noeud.

La sauvegarde de points de reprise consiste à enregistrer une image instantanée de l'état actuel de l'application. Elle peut être utilisée par la suite pour le redémarrage de l'application en cas de panne. C'est une opération que nous retrouvons aussi bien dans les protocoles coordonnés que dans les protocoles non coordonnés. Nous considérons la sauvegarde de points de reprise fournie dans la librairie BLCR [4] (*Berkeley Lab Checkpoint/Restart*), qui est disponible dans l'implémentation MPICH2 [5]. La sauvegarde de points de reprise avec BLCR est réalisée dans le noyau de Linux et permet de sauvegarder toutes les ressources du noyau.

Pour mesurer la puissance électrique pendant la sauvegarde d'un point de reprise, nous considérons un banc d'essai où un processus sauvegarde un point de reprise d'une taille de 1 Go. Pour sauvegarder un point de reprise de 1 Go, nous utilisons un banc d'essai qui alloue 1 Go de données en mémoire. Par exemple, si on exécute une application de haute performance comme CM1 [6] avec 16 processus et un maillage de 1000x1000x40, chaque processus aura à sauvegarder des points de reprise d'une taille égale à 1 Go. La figure 3.2 présente les mesures de la puissance électrique obtenues, pendant que chaque noeud de la grappe *Sagittaire* sauvegarde un point de reprise de 1 Go sur le disque dur local. La figure 3.3 présente les mesures de la puissance électrique obtenues, pendant que trois noeuds spécifiques de la grappe *Stremi* sauvegardent un point de reprise de 1 Go sur le disque dur local. Les trois noeuds spécifiques sont : le noeud qui consomme le plus, celui qui consomme le moins et celui dont la puissance électrique se situe à la médiane. Sur la figure 3.3,

4. BLCR : https://ftg.lbl.gov/projects/CheckpointRestart/
5. MPICH2 : http://www.mcs.anl.gov/research/projects/mpich2/
6. Cloud Model 1 : http://www.mmm.ucar.edu/people/bryan/cm1/

nous n'avons représenté les puissances électriques que pour ces trois noeuds de la grappe *Stremi* pour permettre une meilleure visibilité.

Il est à noter que *Sagittaire* et *Stremi* sont deux grappes dotées de deux infrastructures différentes de capture de la puissance électrique. Sur l'axe des abscisses, est représentée la durée de la sauvegarde du point de reprise pour tous les noeuds en ce qui concerne la grappe *Sagittaire* et pour trois noeuds spécifiques en ce qui concerne la grappe *Stremi*.

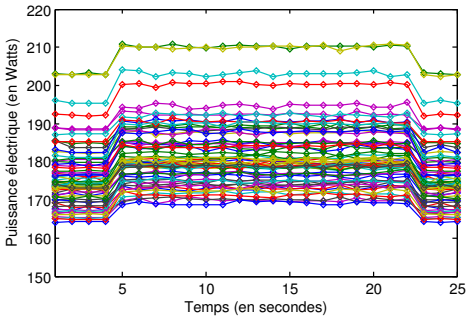

FIGURE 3.2 – Puissance électrique lors d'une sauvegarde d'un point de reprise de 1 Go sur disque pour tous les noeuds de la grappe *Sagittaire*

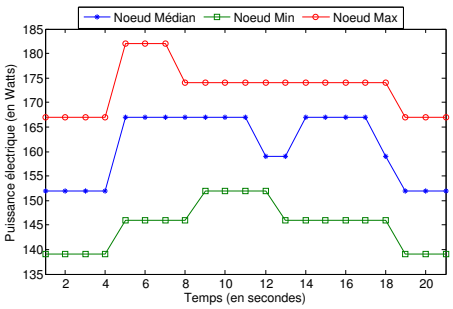

FIGURE 3.3 – Puissance électrique lors d'une sauvegarde d'un point de reprise de 1 Go sur disque pour trois noeuds spécifiques de la grappe *Stremi*

Dans le protocole expérimental considéré, les noeuds de chacune des deux grappes sont d'abord au repos ; l'opération de sauvegarde de point de reprise démarre après 5 secondes d'inactivité. Elle s'arrête à la 22^e seconde pour les noeuds de la grappe *Sagittaire* et à la 18^e seconde pour les noeuds de la grappe *Stremi*. La figure 3.2 montre que la puissance électrique évolue de la même manière pour les mêmes noeuds de la grappe *Sagittaire*. Toutefois, même si les noeuds d'une même grappe sont identiques, leur puissance électrique est différente pendant la période d'inactivité et pendant la phase de sauvegarde du point de reprise. Par ailleurs, nous remarquons sur la figure 3.2 que la puissance électrique pendant la sauvegarde du point de reprise est quasiment constante.

Les mesures électriques obtenues avec la grappe *Stremi* semblent être moins précises étant donné que la fréquence des mesures est basse pour cette grappe de calcul et que la précision des mesures est faible. En effet, les unités de distribution d'électricité (PDUs) utilisées fournissent des mesures par paliers de 6 W ce qui rend les mesures électriques très peu précises surtout lorsque nous mesurons des augmentations de la puissance électrique qui sont de l'ordre de quelques watts. Au delà du fait que leur fréquence de mesure est faible, ces dispositifs de mesure ne sont pas adaptés pour mesurer des opérations qui ne durent que quelques secondes car leur précision est très faible (+/- 6 W). Pour cette raison, nous pouvons affirmer que ces appareils fournissent des mesures avec une perte d'information et donc avec un manque de fiabilité.

Dans notre étude, nous considérons le protocole d'enregistrement de messages [Guermouche et al., 2011], où le processus émetteur enregistre tous les messages qu'il envoie à d'autres processus, et ensuite les processus sauvegardent leurs mémoires de manière non coordonnée. Dans ce protocole, chaque message envoyé est sauvegardé dans un support de stockage qui peut être soit la mémoire RAM soit le disque dur local. Afin de faire durer l'opération d'enregistrement de messages pendant quelques secondes, nous avons instrumenté le code source fourni par Guermouche et al. [Guermouche et al., 2011] afin de répéter autant de fois qu'on le souhaite, la fonction d'enregistrement de messages qui est utilisée chaque fois qu'un processus envoie un message.

Dans notre banc d'essai, le processus émet un message de 1 Go et la fonction d'enregistrement est répétée 10 fois de suite afin d'obtenir un volume total de 10 Go et pour que l'opération puisse durer quelques secondes. Nous avons exécuté le même banc d'essai pour tous les noeuds de la grappe *Sagittaire*. À cause du manque de précision des mesures relevées par les PDUs Raritan, nous ne nous intéressons plus à la grappe *Stremi*. Les figures 3.4 et 3.5 montrent les mesures électriques que nous obtenons pendant que chacun noeud de la grappe *Sagittaire* enregistre des messages soit dans la mémoire RAM (figure 3.4) soit sur le disque dur local (figure 3.5). Sur l'axe des abscisses est représentée la durée de l'enregistrement de messages pour tous les noeuds de la grappe *Sagittaire*.

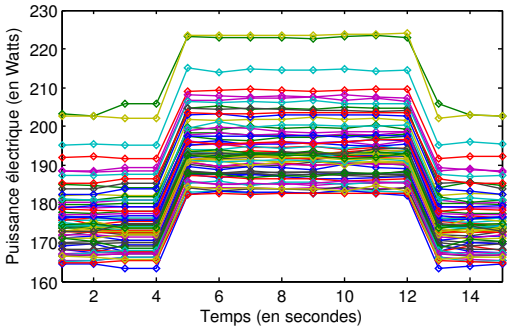

FIGURE 3.4 – Puissance électrique lors de l'enregistrement de messages d'un volume total de 10 Go dans la mémoire RAM pour tous les noeuds de *Sagittaire*

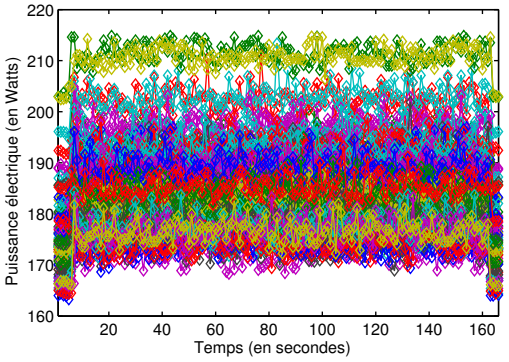

FIGURE 3.5 – Puissance électrique lors de l'enregistrement de messages d'un volume total de 10 Go sur le disque dur local pour tous les noeuds de *Sagittaire*

Dans le protocole expérimental considéré, les noeuds de la grappe *Sagittaire* sont d'abord au repos ; l'enregistrement de messages sur mémoire RAM et sur disque dur local démarrent après 5 secondes d'inactivité. L'enregistrement de messages sur mémoire RAM s'arrête à la 12^e seconde alors que l'enregistrement de message sur disque s'arrête à la 162^e seconde. Sur les figures 3.4 et 3.5, nous observons que la puissance électrique évolue de la même manière pour les mêmes noeuds de la grappe *Sagittaire*. De même, bien que les noeuds d'une même grappe soient identiques,

leur puissance électrique est différente pendant la période d'inactivité et pendant la phase d'enregistrement de messages. De plus, la figure 3.4 montre que la puissance électrique reste constante pendant l'enregistrement de messages sur mémoire RAM. Ceci n'est pas vraiment le cas pour l'enregistrement de messages sur disque dur (voir figure 3.5). Nous constatons également que la puissance électrique lors de l'enregistrement de messages sur mémoire RAM est plus élevée que celle de l'enregistrement de messages sur disque dur local.

Dans notre étude, nous avons considéré le protocole coordonné fourni dans MPICH2. Dans ce protocole, la coordination consiste à synchroniser tous les processus pour s'assurer qu'aucun message n'est en cours de transmission. Ceci est matérialisé par une barrière de synchronisation dans l'implémentation de MPICH2.

Afin de faire durer cette coordination quelques secondes, nous avons simulé une coordination infinie à travers une barrière de synchronisation entre les processus de tous les noeuds de la grappe *Sagittaire*. Pour cela, tous les processus à part un seul, exécutent la routine MPI_Barrier. Nous arrêtons cette coordination infinie après 30 secondes. La figure 3.6 montre les mesures de puissance électrique que nous obtenons pendant la coordination de tous les noeuds de la grappe *Sagittaire*.

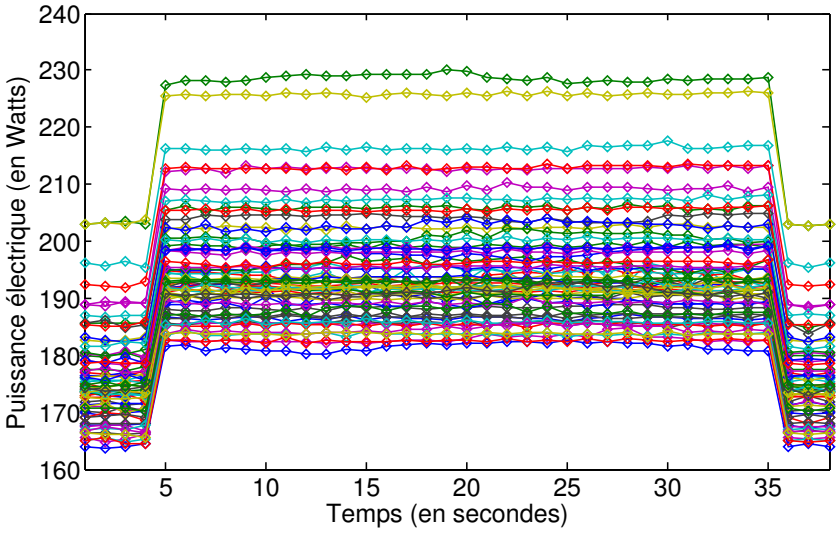

FIGURE 3.6 – Puissance électrique lors de la coordination de tous les noeuds de la grappe *Sagittaire*

Dans le protocole expérimental considéré, les noeuds de la grappe *Sagittaire* sont d'abord au repos ; la coordination démarre après 5 secondes d'inactivité. Elle s'arrête donc à la 35^e seconde. La figure 3.6 montre que la puissance électrique pendant la coordination est quasiment constante. Nous remarquons également que la puissance électrique est très élevée. En effet, pendant la coordination liée à l'implémentation du protocole coordonné considéré, un coeur de chaque noeud de calcul est constamment sollicité étant donné qu'il reste en attente active pour une synchronisation avec

les autres processus appartenant aux autres noeuds. Ce qui est important dans la mesure de la consommation énergétique de la coordination c'est le temps d'exécution de l'opération. Il est équivalent à la durée pendant laquelle les processus restent en train de s'attendre activement. Ainsi, si les processus sont mal synchronisés, la consommation énergétique de leur coordination sera très importante.

3.1.3 Puissance électrique dans les algorithmes de diffusion de données

Dans cette section, nous étudions la puissance électrique de quatre algorithmes de diffusion de données, deux algorithmes utilisés dans MPI et deux algorithmes hybrides (MPI + OpenMP), pour tous les noeuds de la grappe *Taurus* qui est dotée de la même infrastructure de capture de la puissance électrique que la grappe *Sagittaire*. Le choix de cette grappe plus récente permet notamment de considérer plusieurs coeurs par noeud.

Pour les deux algorithmes utilisés dans MPI, nous considérons l'algorithme *Scatter & AllGather* [Thakur and Gropp, 2003] fourni dans l'implémentation MPICH2 [7] 1.3.3 et l'algorithme *Pipelining* fourni dans l'implémentation OpenMPI [8] 1.4.4. Pour plus de commodité, nous choisissons de les appeler respectivement *MPI/SAG* et *MPI/Pipeline*.

Dans un algorithme hybride de diffusion de données, le processus initiateur utilise MPI pour diffuser la donnée à un seul processus MPI par noeud. Ensuite, chaque processus MPI utilise OpenMP pour partager la donnée diffusée avec tous les autres processus qui se trouvent dans le même noeud. Nous appelons *Hybrid/SAG* la diffusion hybride utilisant*MPI/SAG* associée à OpenMP. De façon analogue, *Hybrid/-Pipeline* combine *MPI/Pipeline* avec OpenMP.

Pour les mesures électriques, nous utilisons soit 6 coeurs soit 12 coeurs par noeud pour la diffusion de données sur tous les noeuds de la grappe *Taurus*. Un seul processus diffuse un grand volume de données à tous les autres processus. Afin de faire durer l'opération quelques secondes, nous mesurons une diffusion de 2 Go de données avec chacun des quatre algorithmes.

Nous avons remarqué de nouveau que la puissance électrique pendant une diffusion de données n'est pas la même pour tous les noeuds identiques de la grappe. Pour une meilleure visibilité, nous affichons sur les figures 3.7 et 3.8, les mesures électriques moyennes sur tous les noeuds. Les barres autour de chaque point représentent les écarts de puissance électrique entre le noeud qui consomme le plus et le noeud qui consomme le moins. Les figures 3.7 et 3.8 montrent respectivement les puissances électriques des quatre algorithmes de diffusion de données lorsque nous considérons 6 ou 12 processus par noeud (1 processus sur chaque coeur).

Sur les figures 3.7 et 3.8 nous pouvons relever les observations suivantes :

– Sur le profil électrique de la diffusion de données avec l'algorithme *MPI/SAG*, nous pouvons identifier les deux différentes parties qui correspondent aux deux

7. http://www.mcs.anl.gov/research/projects/mpich2/
8. http://www.open-mpi.org/

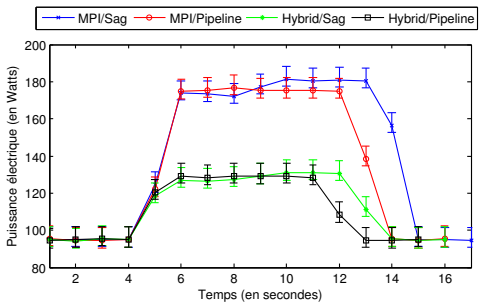

FIGURE 3.7 – Puissance électrique pendant les quatre algorithmes de diffusion de données pour la grappe *Taurus* en utilisant 6 coeurs par noeud

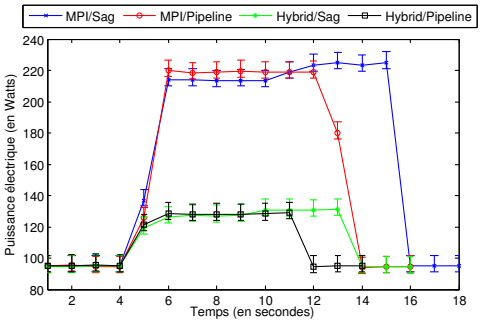

FIGURE 3.8 – Puissance électrique pendant les quatre algorithmes de diffusion de données pour la grappe *Taurus* en utilisant 12 coeurs par noeud

différentes opérations de base utilisées : *Scatter* d'une part et *AllGather* d'autre part.

- La puissance électrique pendant la diffusion de données avec l'algorithme *MPI/Pipeline* est quasiment constante, ce qui correspond à une unique opération de base qui est l'opération *Pipeline*.
- La puissance électrique lors de la première opération de base de l'algorithme *MPI/SAG* est légèrement plus faible que celle de la diffusion de données avec *MPI/Pipeline*.
- La puissance électrique lors de la seconde opération de base de l'algorithme *MPI/SAG* est légèrement plus élevée que celle de la diffusion de données avec *MPI/Pipeline*.
- Les diffusions hybrides avec *Hybrid/SAG* ou *Hybrid/Pipeline* ont des profils temporels de puissance électrique respectivement semblables à *MPI/SAG* ou *MPI/Pipeline*.
- La puissance électrique lors des diffusions hybrides est beaucoup plus faible que celle des algorithmes utilisés dans MPI. En effet, dans une diffusion hybride, les opérations MPI impliquent seulement un coeur par noeud étant donné que seul un processus MPI par noeud est actif pendant ces étapes de la diffusion de données.
- Quel que soit l'algorithme de diffusion de données, nous observons une différence importante entre la puissance électrique avec 6 processus par noeud (*i.e.,* 6 coeurs par noeud) ou 12 processus par noeud (*i.e.,* 12 coeurs par noeud).
- La durée de la diffusion de données varie en fonction de l'algorithme et du nombre de processus par noeud. Pour une diffusion hybride, le nombre de processus par noeud influe très peu sur la durée des opérations de diffusion de données liées aux opérations MPI.
- Bien que les noeuds soient identiques d'un point de vue matériel, leur puissance électrique est différente pour un même algorithme de diffusion de données.

Cependant, les différences de puissance électrique entre les noeuds sont moins conséquentes que celles que nous avons relevées dans les grappes *Sagittaire* et *Stremi*.

3.1.4 Analyse des mesures électriques

Afin de consolider nos résultats, nous confrontons les mesures électriques obtenues précédemment à celles que nous obtenons pour différents bancs d'essai qui utilisent intensément une ressource spécifique (processeur, disque dur et mémoire RAM).

À ce titre, nous avons sélectionné les bancs d'essai suivants :

- `idle` : Ce banc d'essai consiste à ne rien lancer sur la machine. La machine est donc allumée et n'exécute rien (à l'exception du système d'exploitation). Elle se trouve donc dans un état d'inactivité.
- `iperf` [9] : Cet outil génère des communications intensives sur le réseau dans l'optique de mesurer son débit. Il peut être utilisé avec un trafic TCP ou UDP. Pour lancer un test `iperf`, on définit un serveur et un client. Dans nos mesures, nous utilisons cet outil avec un trafic TCP.
- `hdparm` [10] : Cette application fournit une interface en ligne de commande pour divers noyaux supportés par le sous-système SATA/PATA/SAS *libATA* de Linux. Nous utilisons l'option `-t` afin d'utiliser intensément le disque dur.
- `cpuburn` [11] : Ce banc d'essai chauffe n'importe quel CPU à la température de fonctionnement maximale que l'on peut atteindre en exécutant un grand volume d'opérations.
- `burnMMX` [12] : Ce programme, inclus dans le paquet de `cpuburn`, utilise intensément le cache et la mémoire.

Dans une première expérience, nous avons exécuté successivement 15 secondes de `idle`, 15 secondes de `cpuburn` sur 1 coeur, 15 secondes de `hdparm` sur 1 coeur, 15 secondes de `burnMMX` sur 1 coeur, et ce sur tous les noeuds de la grappe *Sagittaire*. Nous affichons sur la figure 3.9 la puissance électrique mesurée pour trois noeuds typiques de la grappe *Sagittaire* lors de l'exécution des quatre bancs d'essai considérés. Les trois noeuds typiques sont : le noeud qui consomme le plus, celui qui consomme le moins et celui dont la puissance électrique se situe à la médiane.

Nous observons que la puissance électrique reste constante pendant l'exécution de `cpuburn`. Nous remarquons également que les puissances électriques lors de `cpuburn` sont voisines de celles que nous avons mesurées pour la coordination (voir figure 3.6). En effet, pendant une coordination, les processus les plus rapides restent en attente active pour des messages de synchronisation provenant des processus les plus lents. De même, nous notons que les puissances électriques lors de `hdparm` sont voisines de celles que nous observons pour la sauvegarde des points de reprise ou

9. `iperf` : http://iperf.fr
10. `hdparm` : http://linux.die.net/man/8/hdparm
11. `cpuburn` : http://manpages.ubuntu.com/manpages/precise/man1/cpuburn.1.html
12. `burnMMX` : http://pl.digipedia.org/man/doc/view/burnMMX.1

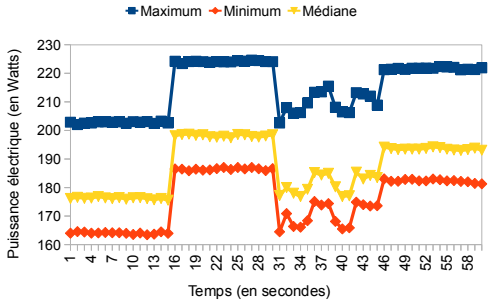

FIGURE 3.9 – Puissance électrique lors des différents bancs d'essai utilisant intensément différentes ressources pour trois noeuds typiques de la grappe *Sagittaire*

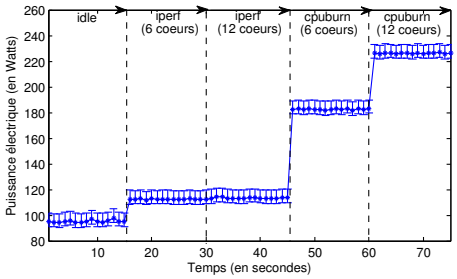

FIGURE 3.10 – Puissance électrique lors des différents bancs d'essai utilisant intensément différentes ressources pour trois noeuds typiques de la grappe *Taurus*

l'enregistrement de messages sur disque dur local. De plus, nous observons que les puissances électriques lors de `burnMMX` sont voisines de celles que nous obtenons pour l'enregistrement de messages dans la mémoire RAM.

Dans une deuxième expérience, nous avons exécuté successivement 15 secondes de `idle`, 15 secondes de `iperf` sur 6 coeurs, 15 secondes de `iperf` sur 12 coeurs, 15 secondes de `cpuburn` sur 6 coeurs, et 15 secondes de `cpuburn`sur 12 coeurs, et ce sur tous les noeuds de la grappe *Taurus*. Nous affichons sur la figure 3.10 la puissance électrique que nous obtenons pour le noeud médian de la grappe *Taurus* lors de l'exécution des cinq bancs d'essai considérés. La figure présente également les écarts de la puissance électrique médiane par rapport au noeud qui consomme le moins et celui qui consomme le plus.

Contrairement à `cpuburn`, nous remarquons que la puissance électrique lors de l'exécution de `iperf` ne varie pas avec le nombre de coeurs qui exécutent ce banc d'essai. De plus, nous notons que les puissances électriques lors des opérations liées à la diffusion de données sont plus proches que de celles du banc d'essai `cpuburn` exécuté sur un même nombre de coeurs. Ceci s'explique par le fait que les coeurs de calcul des noeuds impliqués dans la diffusion de données sont intensément actifs afin d'atteindre les meilleurs performances.

La première section de ce chapitre montre plusieurs résultats préliminaires. Premièrement, nous avons remarqué dès la mesure de la puissance électrique pendant la sauvegarde de points de reprise qu'il est indispensable d'avoir à sa disposition des équipements de mesures suffisamment précis pour observer des fluctuations électriques de quelques watts. En effet, du fait de leur grande incertitude de mesure, les unités de distribution électriques Raritan ne permettent pas de relever de manière fiable de faibles variations par rapport à la puissance électrique au repos.

Deuxièmement, que ce soit pour les protocoles de tolérance aux pannes ou pour les algorithmes de diffusion de données, nous constatons que lors d'une opération de base, la puissance électrique reste quasiment constante. Ce résultat nous amène à

nous poser la question de savoir si cette évolution stationnaire de la puissance électrique lors d'une opération de base n'est pas seulement liée au fait que nos wattmètres n'ont pas une fréquence de mesure suffisamment élevée, c'est-à-dire supérieure à 1 mesure par seconde.

Troisièmement, que ce soit pour les protocoles de tolérance aux pannes ou pour les algorithmes de diffusion de données, nous remarquons que la puissance électrique de noeuds identiques exécutant la même tâche n'est pas la même. Les différences de puissance électrique sont plus ou moins conséquentes d'une grappe de noeuds à l'autre.

Quatrièmement, grâce aux résultats liés aux algorithmes de diffusion de données, nous avons pu noter que la puissance électrique d'un noeud utilisant 12 processus est bien plus importante que celle d'un noeud utilisant seulement 6 processus. Le nombre de processus utilisés par noeud est donc un paramètre qui peut faire varier la puissance électrique consommée par le noeud.

Enfin, nous avons pu noter que les puissances électriques pendant les opérations de base identifiées sont très voisines des mesures électriques que nous obtenons pour différents bancs d'essai qui utilisent intensément les mêmes composants du noeud. Pour la suite de ce chapitre, nous allons donc nous focaliser sur ces bancs d'essai.

3.2 Quel équipement choisir pour mesurer la puissance électrique ?

Dans la deuxième partie de ce chapitre, nous cherchons à choisir l'équipement le plus adéquat pour mesurer la puissance électrique à l'échelle d'une application. Pour répondre à cette question, nous comparons différents appareils de mesure de la puissance électrique. Pour cela, nous relevons une série de mesures de puissance électrique relevées à l'aide de différents dispositifs lors de l'exécution de différents bancs d'essai sur deux systèmes informatiques distincts : un serveur et un ordinateur de bureau. Pour cette étude, nous utilisons les bancs d'essai que nous avons utilisés dans la section 3.1.4. Nous avons considéré uniquement des équipements de mesure qui présentent dans leurs caractéristiques techniques une précision relativement élevée et une fréquence de mesure supérieure à une mesure par seconde. Pour évaluer et analyser successivement la consommations énergétique, la variabilité et l'évolution de la puissance électrique, nous nous appuyons sur la même série de mesures.

La section 3.2.1 présente le dispositif expérimental utilisé. La section 3.2.2 présente une analyse des consommations énergétiques calculées à partir des différents appareils de mesure de la puissance électrique. Dans la section 3.2.3, nous analysons la variabilité des mesures de puissance électrique de ces différents équipements de mesure. Dans la section 3.2.4, nous étudions les profils temporels de puissance électrique obtenus avec les différents wattmètres externes et internes simultanément, en faisant varier la fréquence de mesure et en examinant finement les mesures relevées sur les voies électriques internes. La section 3.2.5 présente une discussion sur les

observations et résultats que nous avons pu relever.

3.2.1 Dispositif expérimental

Cette section décrit les appareils de mesure électrique, l'environnement logiciel pour l'acquisition des mesures électriques, les plates-formes cibles et les bancs d'essai (i.e. *benchmarks*) utilisés dans notre évaluation.

3.2.1.1 Equipements de mesure électrique

Nous classons les appareils de mesure en deux catégories :

- d'une part, les wattmètres externes en courant alternatif qui sont directement branchés à la prise électrique qui sert à alimenter tout l'ordinateur ;
- d'autre part, les wattmètres internes en courant continu, chargés de mesurer les différents câbles ATX issus du bloc d'alimentation qui servent à alimenter les différents composants de l'ordinateur (carte mère, disques durs, lecteur CD-ROM, etc.).

Les wattmètres externes que nous utilisons, mesurent la puissance électrique active (ou moyenne) qui est obtenue en divisant par une période de temps T (ici la seconde) l'énergie consommée pendant T. L'énergie consommée est approximée par une intégration sur un nombre très grand (plusieurs centaines par seconde) de mesures de la puissance instantanée. Les wattmètres internes mesurent les puissances électriques instantanées des lignes ATX servant à alimenter les différents composants de la machine. Étant donné que ces wattmètres mesurent un courant continu et que la tension aux bornes de chaque ligne ATX est connue (3.3 V, 5 V ou 12 V), ces wattmètres n'ont besoin de mesurer que le courant électrique qui alimente chacune des lignes. La puissance instantanée est obtenue en multipliant le courant par la tension aux bornes de la ligne ATX.

Le tableau 3.2 présente en détail les spécifications des wattmètres que nous utilisons. La première ligne du tableau présente les noms que nous utilisons pour appeler les différents wattmètres. La deuxième ligne indique l'entreprise qui a fabriqué chacun de ces wattmètres. Les wattmètres externes mesurent la puissance électrique moyenne à partir du câble d'alimentation de la machine. Les wattmètres internes mesurent la puissance électrique instantanée à partir des lignes électriques ATX qui sortent du bloc d'alimentation de la machine. Le wattmètre OmegaWatt utilisé peut permettre de mesurer les puissances électriques moyennes de 6 machines simultanément. En ce qui concerne les mesures présentées dans cette section, nous ne l'avons utilisé que pour mesurer une seule prise électrique à la fois. Nous n'avons pas d'information sur les capteurs utilisés pour mesurer la puissance électrique avec OmegaWatt et WattsUp.

Nous nous sommes focalisés sur les appareils de mesure de la puissance électrique mais il existe d'autres moyens pour relever la puissance électrique, tels que IPMI (*Intelligent Platform Management Interface*) et les compteurs RAPL (*Running Average Power Limit*). Ces capteurs internes sont des techniques prometteuses pour la mesure de la puissance électrique dans des systèmes distribués à large échelle dans

	Externes AC		Internes DC		
Wattmètres	OmegaWatt	WattsUp	PowerMon2 [Bedard et al., 2010]	NI	DCM
Entreprise	OmegaWatt [a]	WattsUp ? [b]	RENCI iLab [c]	National Instruments [d]	Universitat Jaume I
Nature des voies mesurées	câble d'alimentation d'ordinateur	câble d'alimentation d'ordinateur	toutes les lignes ATX (3.3 V, 5 V et 12 V)	les lignes ATX 12 V	les lignes ATX 12 V
Nombre de voies pouvant être mesurées	6	1	8	32	12
Type de mesure	puissance moyenne	puissance moyenne	puissance instantanée	puissance instantanée	puissance instantanée
Capteurs de puissance	-	-	résistances *Analog Devices ADM1191*	transducteurs *LEM HXS 20-NP*	transducteurs *LEM HXS 20-NP*
Fréquence de mesure (Hz) par canal	1	1	100	1000	28
Précision	±1%	±1.5%	±5%	±1%	±1%
Interface	RS232	USB	USB	USB	RS232
Prix	600 € pour les 6 prises	200 €	125 €	2700 €	Non commercialisé

a. OmegaWatt : http://www.omegawatt.fr/
b. WattsUp : https://www.wattsupmeters.com/
c. PowerMon2 : http://ilab.renci.org/powermon
d. NI : http://www.ni.com/

TABLE 3.2 – Spécifications des wattmètres

la mesure où elles ne requièrent aucun dispositif supplémentaire. Cependant, elles restent non adaptées pour le moment puisqu'elles fournissent des mesures de puissance électrique à des fréquences relativement faibles et que ces mesures ne sont pas suffisamment précises.

3.2.1.2 Plates-formes cibles

L'analyse et l'évaluation réalisée par les wattmètres ont été effectués sur deux plates-formes différentes : un ordinateur de bureau et un serveur. L'ordinateur de bureau, appelé Intel_Desktop, se compose d'un processeur Intel Ivy Bridge Core i7-3770K équipé de 4 coeurs de calcul cadencés à 3.50 GHz et 16 Go de mémoire RAM. La machine serveur, appelée AMD_Server, intègre 4 AMD Opteron 6172 de 12 coeurs de calcul (soit un total de 48 coeurs) cadencés à 2.10 GHz et 256 Go de mémoire RAM.

3.2.1.3 Bancs d'essai

Pour évaluer le comportement de la puissance électrique et de la consommation énergétique, nous exécutons différents programmes qui utilisent intensément des composants spécifiques des plates-formes. Le processeur (CPU), la mémoire, les cartes réseaux, et les disques durs sont les principaux composants que nous sollicitons intensément dans nos expérimentations. Nous utilisons les mêmes bancs d'essai que

dans la section 3.1.4. Nous exécutons les processus `cpuburn` et `burnMMX` sur des coeurs de calcul donnés dans le but de mesurer la puissance électrique avec un nombre variable de coeurs utilisés.

3.2.2 Analyse de la consommation énergétique

Dans cette section, nous analysons la variabilité et la précision des wattmètres externes et internes en utilisant les bancs d'essai présentés précédemment sur les deux machines choisies INTEL_DESKTOP et AMD_SERVER.

La figure 3.11 présente la consommation énergétique mesurée pendant 60 secondes d'exécution de chacun des bancs d'essai sur les deux plates-formes. Les histogrammes étiquetés `idle` représentent la consommation énergétique lorsque la plate-forme ne fait rien pendant une minute complète. Les histogrammes `hdparm` et `iperf` montrent l'énergie enregistrée par les wattmètres lorsque les disques durs et les cartes réseaux sont respectivement utilisés intensément. Pour les bancs d'essai `burnMMX` et `cpuburn`, nous chauffons tous les coeurs de calcul en exécutant un processus par coeur. Plus précisément, nous utilisons la commande `taskset` pour lier les processus aux coeurs de calcul des deux machines. Pour les wattmètres internes, nous représentons la consommation énergétique agrégée calculée en ajoutant les mesures énergétiques de toutes les lignes 12 V. Même si POWERMON2 mesure également les lignes 3.3 V et 5 V [Bedard et al., 2010], nous préférons ne considérer que les lignes 12 V, dans l'optique de fournir une comparaison juste avec les autres wattmètres internes qui ne mesurent que les lignes 12 V.

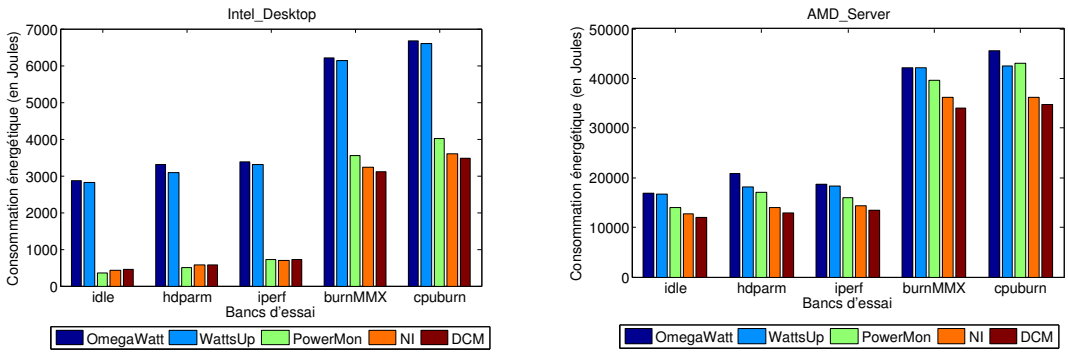

FIGURE 3.11 – Consommation énergétique des bancs d'essai mesurée à l'aide des différents wattmètres.

La figure 3.11 montre que les consommations énergétiques relevées par les deux wattmètres externes (OMEGAWATT et WATTSUP) sont vraiment similaires. Cependant, nous observons un comportement différent pour les wattmètres internes. En effet, dans l'énergie mesurée par ces équipements, il est facile d'observer que les valeurs relevées avec POWERMON2 sont presque toujours plus élevées que celles que nous avons relevées à l'aide de NI et DCM. Ces variations sont principalement dues à l'utilisation de composants différents pour mesurer le courant électrique des câbles

internes. En effet, NI et DCM utilisent les mêmes capteurs de puissance électrique contrairement à ceux qui sont utilisés par PowerMon2.

Contrairement aux wattmètres externes, les équipements internes mesurent la puissance électrique en aval (à la sortie) du bloc d'alimentation. Par conséquent, les mesures internes ne prennent pas en compte la puissance électrique consommée par le bloc d'alimentation et d'autres composants comme les disques durs. Ceci explique pourquoi la consommation énergétique relevée par les wattmètres internes est plus faible que celle qui est enregistrée par les wattmètres externes. Il est possible d'observer cela sur la figure 3.11 pour tous les bancs d'essai quelle que soit la plate-forme cible. Fournir une mesure interne en plus d'une valeur externe permet de nous donner une indication sur le rendement du bloc d'alimentation électrique qui peut être différent d'un noeud à l'autre.

Par ailleurs, nous notons que la consommation énergétique de AMD_Server est plus élevée que celle de Intel_Desktop, et ce parce que AMD_Server comporte 48 coeurs alors que Intel_Desktop ne comporte que 4 coeurs. En comparaison à Intel_Desktop, la différence entre les mesures externes et internes semble être relativement moins importante pour AMD_Server. Ceci est notamment dû au fait que AMD_Server est constitué de 48 coeurs de calcul et de plusieurs ventilateurs dont la puissance électrique est incluse dans les mesures internes et représente une part très importante de la consommation totale de la machine.

3.2.3 Analyse de la puissance électrique

Dans cette section, nous analysons les mesures de puissance électrique effectuées par les différents wattmètres étudiés. À travers cette analyse, nous visons d'abord à montrer la variabilité des mesures électriques pour différents bancs d'essai exécutés sur les deux plates-formes cibles : Intel_Desktop et AMD_Server. En comparant les différents wattmètres et plates-formes cibles, nous cherchons à comparer les mesures externes aux valeurs internes avec une fréquence de mesure variable.

Pour cela, nous considérons les scénarios suivants : la machine cible est `idle` (Figure 3.12), un coeur de calcul exécute `hdparm` (figure 3.13), et tous les coeurs de calcul exécutent `cpuburn` (Figure 3.14). Nous présentons seulement les mesures de puissance électrique pour ces trois bancs d'essai car nous avons constaté que `iperf` et `burnMMX` ont respectivement des comportements électriques très analogues aux bancs d'essai `idle` et `cpuburn`. Nous exécutons chaque banc d'essai sur les deux différentes machines cibles et mesurons la puissance électrique pendant 60 secondes en utilisant les wattmètres externes OmegaWatt et WattsUp, et les wattmètres internes PowerMon2, NI et DCM.

Les figures 3.12, 3.13 et 3.14 présentent les *boîtes à moustaches* qui affichent la répartition des mesures de puissance électrique prélevées durant 60 secondes. Le nombre de points affichés dans chaque boîte à moustache dépend de la fréquence des wattmètres. Chaque *boîte à moustaches* permet d'afficher graphiquement les mesures de puissance électrique en les résumant à l'aide de 5 propriétés statistiques : la valeur minimale, le premier quartile (Q1), la médiane (Q2), le troisième quartile

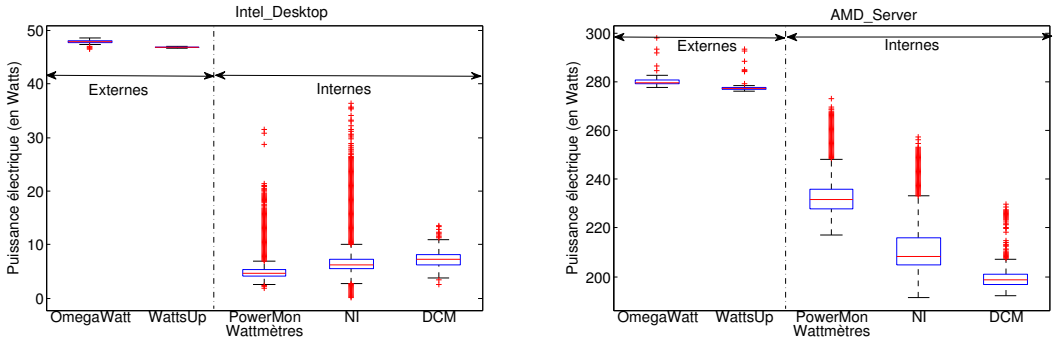

FIGURE 3.12 – Variabilité dans le temps des mesures de puissance électrique pour le banc d'essai `idle`.

(Q3) et la valeur maximale. Les diagrammes permettent aussi de repérer quelles mesures devraient être considérées comme valeurs aberrantes. Une valeur est considérée comme aberrante lorsqu'elle est trop éloignée par rapport aux autres valeurs. Étant donné que les mesures internes ne prennent pas en compte la puissance électrique du bloc d'alimentation et de quelques autres composants, les *boîtes à moustaches* correspondantes sont dans la plupart des cas en dessous de celles qui sont relatives aux wattmètres externes.

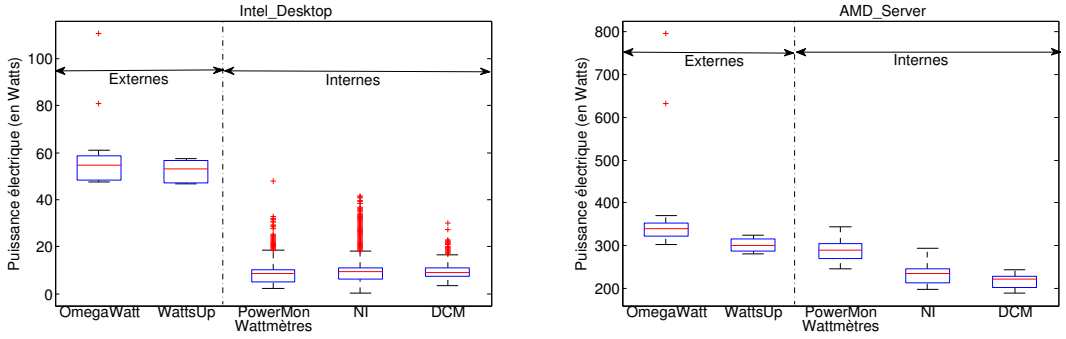

FIGURE 3.13 – Variabilité des mesures de puissance électrique pour le banc d'essai `hdparm`.

Les *boîtes à moustaches* pour INTEL_DESKTOP montrent que les mesures de puissance électrique obtenues avec les équipements externes sont similaires, quel que soit le banc d'essai considéré. Ceci est aussi valable pour les mesures électriques internes à l'exception de `cpuburn` pour lequel POWERMON2 fournit des mesures électriques sensiblement supérieures à celles relevées avec les autres dispositifs internes.

En ce qui concerne AMD_SERVER, des différences apparaissent entre les appareils de mesure externes et internes, particulièrement pour le banc d'essai `cpuburn`. En effet, pour tous les bancs d'essai exécutés sur AMD_SERVER, les mesures élec-

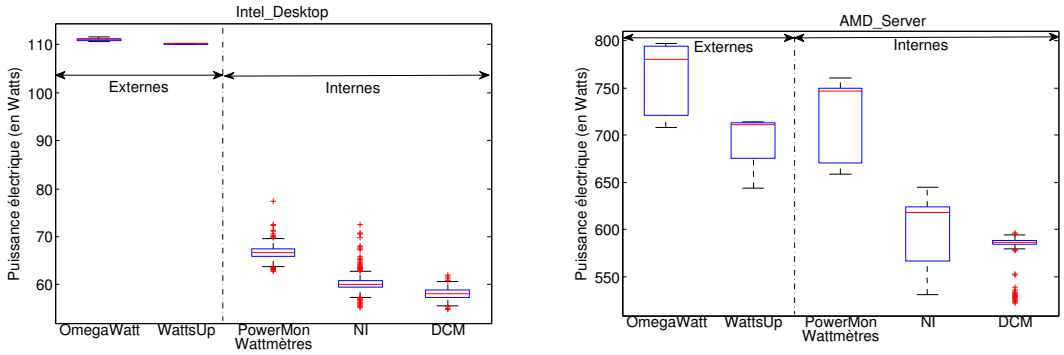

FIGURE 3.14 – Variabilité des mesures de puissance électrique pour le banc d'essai `cpuburn`.

triques obtenues à l'aide de POWERMON2 sont toujours plus élevées que celles qui sont fournies par les autres wattmètres internes. De plus, comparé à WATTSUP, OMEGAWATT fournit des valeurs plus élevées de puissance électrique, spécialement avec les bancs d'essai `cpuburn` et `hdparm`. Ces différences semblent être liées au fait que ces wattmètres sont constitués de différents composants offrant différents degrés de précision, et ce spécialement lorsqu'ils sont utilisés pour mesurer des puissances électriques très élevées (plus que 300W pour AMD_SERVER dans les figures 3.13 et 3.14).

Plus étonnant encore, lorsque `cpuburn` est exécuté sur AMD_SERVER, les mesures de puissance électrique relevées par POWERMON2 sont plus élevées que celles de WATTSUP, et ce, même si les mesures de POWERMON2 ne prennent pas en compte le bloc d'alimentation et certains autres composants internes. Ceci est aussi le cas pour certaines mesures électriques lorsque le banc d'essai `hdparm` est exécuté sur la machine AMD_SERVER.

Par ailleurs, particulièrement sur la figure 3.12, nous remarquons qu'à l'opposé des mesures externes, les équipements internes mesurent des valeurs plus étalées et génèrent plusieurs valeurs aberrantes. Ceci est très certainement dû à la fréquence de mesure des appareils internes qui est très élevée et permet ainsi de mesurer certaines fluctuations de puissance que les équipements externes sont incapables de capter. De plus, la figure 3.14 montre que lorsque `cpuburn` s'éxécute sur AMD_SERVER, les mesures électriques sont fortement étalées : une étendue de presque 100 W pour OMEGAWATT, POWERMON2, NI ; et d'environ 70 W pour WATTSUP et DCM. Cette grande étendue des mesures électriques pour `cpuburn` sur AMD_SERVER montre que ce banc d'essai fluctue beaucoup. Nous analysons ce phénomène dans la section suivante en examinant les profils de puissance électrique dans le temps.

3.2.4 Analyse des profils temporels de puissance électrique

Dans cette section, nous poursuivons notre analyse du comportement des bancs d'essai sur les machines cibles. D'abord, nous étudions les profils temporels de puis-

sance électrique. Ensuite, nous examinons l'impact de la fréquence de mesure des wattmètres sur les profils temporels. Finalement, nous analysons les puissances électriques alimentant chaque câble interne à l'aide de PowerMon2 particulièrement afin de détecter d'où proviennent les fluctuations de la puissance électrique.

3.2.4.1 Mesures électriques simultanées : externes vs internes

Considérons les figures 3.15 et 3.16 qui, respectivement, représentent les profils électriques mesurés sur Intel_Desktop et AMD_Server. Ces deux figures affichent les mesures électriques correspondant à 20 secondes d'exécution des bancs d'essai `idle`, `hdparm`, et `cpuburn` obtenues à la fois par les wattmètres externes et internes.

Sur la figure 3.15, le graphique représenté à gauche montre l'évolution de la puissance électrique lorsque OmegaWatt et PowerMon2 (mesurant seulement les lignes de 12V) sont simultanément branchés à Intel_Desktop ; le graphique représenté à droite affiche la même information mais avec WattsUp et NI. Le but de cette comparaison est de reproduire le même scénario avec deux dispositifs : l'un mesurant à l'extérieur et l'autre mesurant en interne. Pour cette étude, nous nous intéressons à PowerMon2 et à NI car ce sont les deux wattmètres internes qui ont les fréquences de mesure les plus élevées.

Les résultats montrent que les profils électriques issus des wattmètres externes sont quasiment identiques mais qu'il existe quelques différences entre les mesures provenant des appareils internes. En outre, nous observons beaucoup plus de fluctuations avec les wattmètres internes, particulièrement avec NI. Ce comportement est lié à la grande fréquence de mesure de ces équipements. Ces comparaisons permettent de montrer aisément si différents wattmètres (externes *vs* internes) fournissent des profils électriques fiables pour un même scénario.

Les mesures de la puissance électrique peuvent être décalées dans le temps dans la mesure où il n'y a pas une synchronisation parfaite entre le début de la mesure et le début de l'exécution du banc d'essai. En affichant les profils électriques provenant de wattmètres internes, notre objectif est d'analyser le comportement interne des applications ; par exemple, pour détecter des variations rapides qui peuvent être lissées par les wattmètres externes car leur fréquence de mesure plus faible entraine un effet de moyenne.

La figure 3.16 permet de noter dans quelle mesure les profils électriques issus des wattmètres externes sont différents pour `cpuburn`. Ces différences proviennent principalement des caractéristiques des appareils et des potentiels facteurs environnementaux (comme la température de la salle d'expérimentation), étant donné que ces expériences ne pouvaient pas être réalisées simultanément. La même observation est notable pour `cpuburn` avec les wattmètres internes où le profil électrique obtenu avec PowerMon2 est fortement décalé dans le temps de celui que nous obtenons à l'aide de NI.

Nous observons également les pics et creux observés dans le profil électrique du banc d'essai `cpuburn` lorsqu'il tourne sur la machine AMD_Server. Rappelons

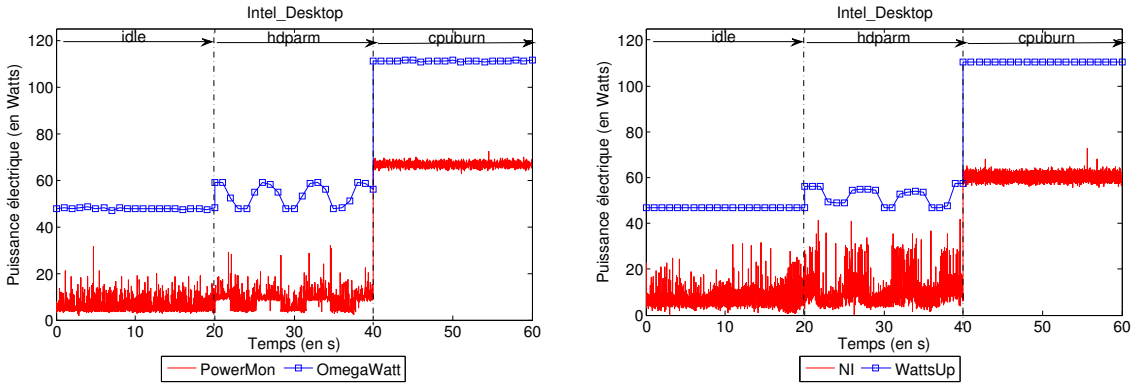

FIGURE 3.15 – Profils temporels de puissance électrique obtenus lorsque nous exécutons successivement `idle`, `hdparm` et `cpuburn` sur la machine INTEL_DESKTOP.

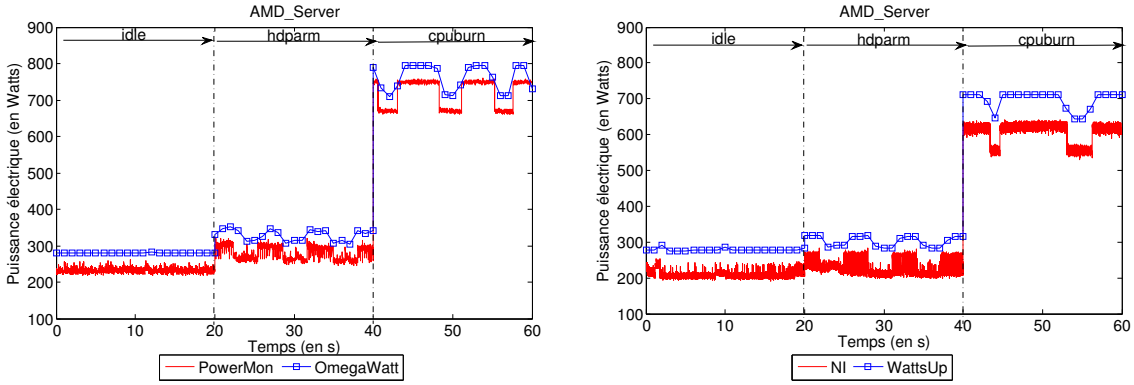

FIGURE 3.16 – Profils temporels de puissance électrique obtenus lorsque nous exécutons successivement `idle`, `hdparm` et `cpuburn` sur la machine AMD_SERVER.

que cette plate-forme est composée de 48 coeurs sur lesquels tournent 1 processus `cpuburn` par coeur. Ce comportement est lié aux ventilateurs qui s'éteignent et s'allument de façon variable afin de refroidir la machine et de maintenir la température de la plate-forme à un niveau constant.

3.2.4.2 Impact de la fréquence de mesure : le cas de NI

Dans cette section, nous nous intéressons à étudier si une fréquence élevée (supérieure à 100 Hz) mesure des fluctuations électriques que l'on ne peut pas détecter avec une fréquence de mesure relativement faible (1 Hz). Pour cela, nous mesurons la puissance électrique avec NI à une fréquence de 1000 Hz pendant 30 secondes d'exécution de `hdparm` et de `cpuburn` sur INTEL_DESKTOP et AMD_SERVER. Ensuite, nous réduisons la fréquence de 1000 Hz à 1 Hz (en passant par les fréquences intermédiaires) en calculant les moyennes des puissances électriques sur des intervalles de temps de plus en plus larges. Par exemple, les puissances électriques réduites à une fréquence de 1 Hz sont obtenues en prenant toutes les 1000 mesures et en calculant les moyennes sur chaque ensemble de 1000 mesures. Pour cela, nous appliquons la

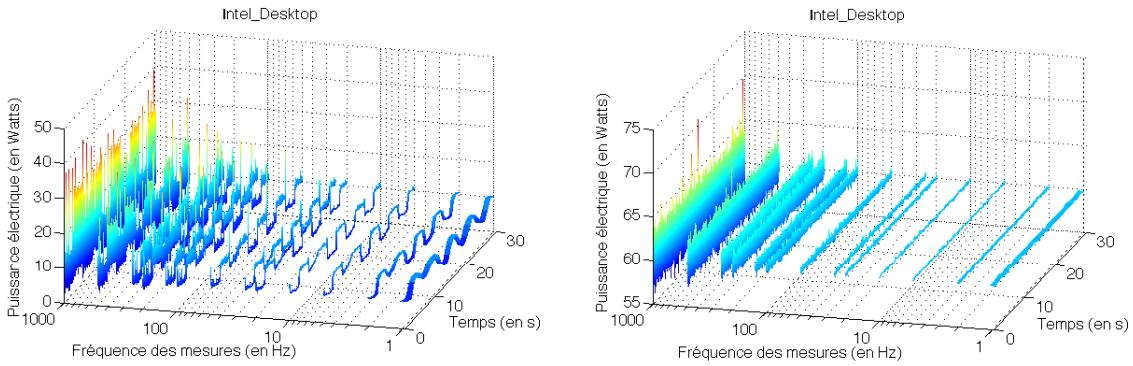

FIGURE 3.17 – Profils électriques obtenus avec NI pour les bancs d'essai `hdparm` (à gauche) et `cpuburn` (à droite) exécutées sur INTEL_DESKTOP avec une fréquence de mesure variable

formule suivante :

$$S_j^r = r\frac{\sum_{i=1+1000(j-1)/r}^{\frac{1000j}{r}}(S_i^{1000})}{1000}, \tag{3.1}$$

où r est la fréquence de mesure après réduction, S_i^{1000} est la i^e mesure prise à 1000 Hz et S_j^r est la j^e mesure obtenue à une fréquence égale à r. j est l'indice de la puissance électrique moyenne avec la fréquence après réduction. Par exemple, S_1^r est la 1^{re} puissance électrique moyenne que nous obtenons à la fréquence r.

La figure 3.17 montre les profils de puissance obtenus avec NI en réduisant la fréquence de mesure de 1000 Hz à 1 Hz (en passant par des valeurs intermédiaires) pour `hdparm` (à gauche) et `cpuburn` (à droite) sur INTEL_DESKTOP. À partir de cette figure, nous constatons que les fluctuations captées par la fréquence de mesure élevée (supérieure à 200 Hz) masque les pics et creux du banc d'essai `hdparm` et les rend plus difficiles à percevoir. Cependant, une fréquence de mesure réduite (inférieure à 50 Hz) dissimule quelques fluctuations électriques intéressantes, comme les hauts pics que nous observons juste avant chaque creux lorsque la fréquence de mesure est à 50 Hz. En ce qui concerne le banc d'essai `cpuburn`, nous pouvons noter que la puissance électrique fluctue entre 57 W et 63 W lorsque la fréquence de mesure est à 1000 Hz. L'allure du profil électrique s'amincit au fur et à mesure que la fréquence de mesure diminue. En dessous de 50 Hz, le profil électrique obtenu est une ligne constante dépourvu de bruit. Ainsi, pour INTEL_DESKTOP, il est plus judicieux de choisir une fréquence de mesure intermédiaire (entre 50 Hz et 200 Hz) pour un banc d'essai comme `hdparm`, tandis qu'une fréquence de mesure relativement faible (*i.e.,* 1 Hz) est amplement suffisante pour le profil électrique d'un banc d'essai tel que `cpuburn`.

La même expérience a été reproduite pour AMD_SERVER et les résultats sont donnés sur la figure 3.18. Dans ce cas, nous notons que le comportement de `hdparm` est similaire à celui que nous avons observé pour INTEL_DESKTOP. Par contre, le comportement du banc d'essai `cpuburn` exécuté sur AMD_SERVER est différent

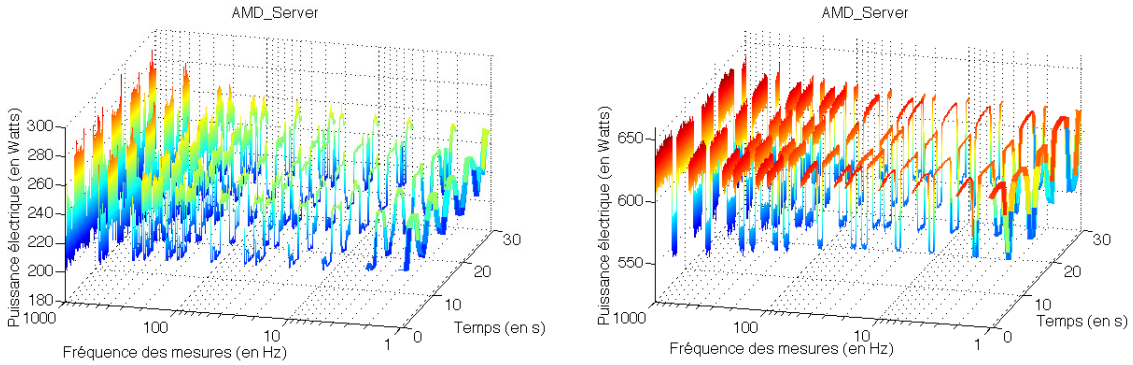

FIGURE 3.18 – Profils électriques obtenus avec NI pour les bancs d'essai `hdparm` (à gauche) et `cpuburn` (à droite) exécutées sur AMD_SERVER avec une fréquence de mesure variable

de celui que nous avons observé sur INTEL_DESKTOP. En effet, une fréquence de mesure intermédiaire aide à mieux comprendre les pics et creux qui apparaissent lorsque `cpuburn` est exécuté sur AMD_SERVER. Avec une fréquence de 1 Hz, nous percevons toujours les pics et creux de `cpuburn`, ce qui confirme que ces fluctuations électriques restent visibles sur AMD_SERVER même sur une échelle de temps plus large (la seconde). Comme expliqué précédemment, ces pics et creux sont dus aux rapides mises en marche et extinctions des ventilateurs de AMD_SERVER.

En somme, mesurer à une fréquence très élevée (plus de 500 Hz) n'est pas toujours indispensable pour l'étude du profil temporel de la puissance électrique d'une application. Cela peut même générer un bruit qui masque l'allure générale du profil électrique. La fréquence de mesure la plus adaptée n'est pas toujours la fréquence la plus élevée, car ce n'est pas elle qui permet de mieux comprendre les fluctuations électriques pouvant se produire pendant l'application exécutée.

3.2.4.3 Analyse des voies électriques internes : le cas de POWERMON2

Dans cette section, nous fournissons une analyse des profils temporels de puissance électrique décrits par les bancs d'essai `idle`, `hdparm`(1 processus `hdparm`par machine), et `cpuburn` (1 processus `cpuburn` par coeur) lorsque POWERMON2 est utilisé pour mesurer indépendamment les lignes 3.3 V, 5 V et 12 V des machines INTEL_DESKTOP et AMD_SERVER. Nous cherchons à identifier pour chacune des deux machines, les composants qu'alimentent ces lignes électriques. La somme de toutes les lignes est aussi incluse dans les résultats affichés.

Nous décrivons le comportement de INTEL_DESKTOP sur la figure 3.19. Dans cette plate-forme, en fonction du banc d'essai, les différentes lignes ont des puissances électriques différentes. Pour `idle`, la puissance électrique alimentant les lignes 3.3 V et 5 V est plus élevée que celle alimentant chacune des lignes 12 V et elle demeure constante pour toutes les lignes. La situation est différente pour `hdparm` : les lignes 5 V fluctuent conjointement avec les lignes 12 V alimentant les réceptacles de processeurs ; alors que les lignes 3.3 V et 12 V alimentant la carte mère présentent

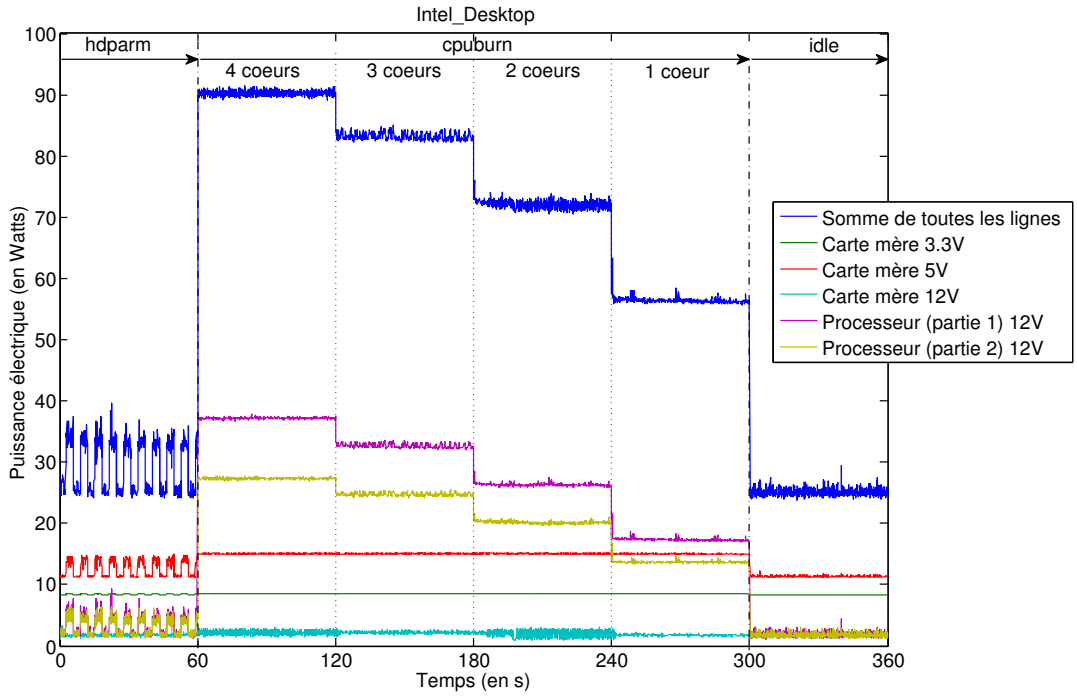

FIGURE 3.19 – Profil temporel de la puissance électrique obtenu à l'aide de POWERMON2 lorsque nous exécutons différents bancs d'essai sur INTEL_DESKTOP.

un profil électrique assez plat. Avec le banc d'essai `cpuburn`, nous observons que les lignes 5 V et les lignes 12 V alimentant les réceptacles de processeurs se caractérisent par une puissance électrique plus élevée, ce qui était prévisible puisque les processus `cpuburn` sollicitent les coeurs de calcul intensément. Il est également important de noter que les lignes de 3.3 V et de 12 V alimentant la carte mère disposent d'un profil électrique très plat.

La même expérience est reproduite sur AMD_SERVER et les résultats sont affichées sur la figure 3.20. Cette plate-forme présente une toute autre gamme de lignes électriques internes que POWERMON2 est capable de mesurer. Dans ce cas, nous analysons les lignes 3.3 V, 5 V et 12 V alimentant la carte mère et les lignes de 12 V alimentant les réceptacles de processeurs. Comme le montre la figure 3.20, chacun des quatre processeurs de AMD_SERVER n'est pas spécifiquement alimenté par une ligne de 12 V correspondant à un réceptacle. En effet, en variant le nombre de processeurs exécutant le banc d'essai `cpuburn` nous observons une variation de la puissance électrique mesurée par chacune des quatre lignes de 12 V alimentant les réceptacles de processeurs. Le banc d'essai `idle` fournit un profil plat pour toutes les lignes ; néanmoins, les lignes de 12 V transportent constamment plus de puissance que les lignes de 3.3 V et de 5 V. La situation diffère pour `hdparm` : les lignes de 12 V débutent par les pics naturels de ce banc d'essai. En particulier, la ligne de 12 V alimentant la carte mère consomme presque deux fois plus que chacune des lignes de

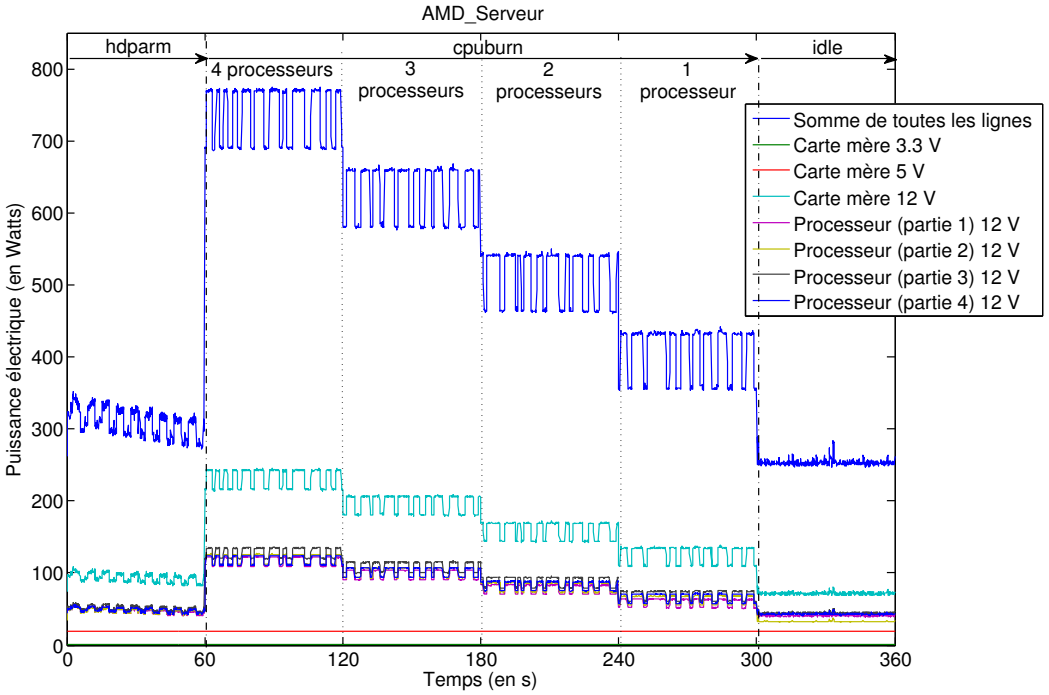

FIGURE 3.20 – Profil temporel de la puissance électrique obtenu à l'aide de POWERMON2 lorsque nous exécutons différents bancs d'essai sur AMD_SERVER.

12 V alimentant les réceptacles. En ce qui concerne l'exécution de `cpuburn`, la situation est semblable : la ligne de 12 V alimentant la carte mère consomme le double de chacune des lignes de 12 V alimentant les réceptacles de processeurs et commence à décroître lorsque nous sollicitons moins de processeurs. Quant aux lignes de 3.3 V et de 5 V alimentant la carte mère, il est intéressant de souligner qu'elles décrivent des profils électriques constants.

3.2.5 Discussion

Dans la deuxième partie de ce chapitre, nous cherchons à choisir l'équipement le plus adapté pour mesurer la puissance électrique d'une application ou d'une opération de cette application. Pour cela, nous avons évalué et analysé différents wattmètres externes et internes lorsqu'ils sont utilisés pour mesurer la puissance électrique durant l'exécution de différents bancs d'essai. Nous avons testé ces équipements sur deux plates-formes différentes, un serveur (AMD_SERVER : 4 processeurs et 12 coeurs chacun) et un ordinateur de bureau (INTEL_DESKTOP : 1 processeur de 4 coeurs), afin de fournir une comparaison complète de ces équipements de mesure de la puissance électrique. Contrairement aux wattmètres externes, les équipements internes mesurent la puissance électrique à la sortie du bloc d'alimentation. Par conséquent, les mesures internes ne prennent pas en compte la puissance électrique du bloc d'alimentation mais aussi d'autres composants comme les disques durs.

D'abord, nous avons montré que contrairement aux wattmètres externes, les équipements internes ne concordent pas en ce qui concerne l'énergie mesurée et la variabilité de la puissance électrique. En effet, les résultats ont montré que les mesures de puissance électrique et de la consommation énergétique relevées par PowerMon2 sont souvent supérieures à celles qui sont fournies par NI et DCM et ce en particulier avec la plate-forme AMD_Server. Nous avons observé jusqu'à 100 W d'écart lorsque ces wattmètres mesuraient la puissance électrique de `cpuburn` s'exécutant sur AMD_Server. Ces résultats s'expliquent par le fait que ces wattmètres utilisent des composants différents pour mesurer la puissance électrique alimentant les câbles internes. Comme on pouvait le prévoir, nous avons montré que l'énergie mesurée par les wattmètres internes est toujours inférieure à celle que nous obtenons avec les wattmètres externes. Toutefois, ceci n'a pas toujours été le cas. En effet, lorsque `cpuburn` est exécuté sur AMD_Server, PowerMon2 relève des mesures de puissance électrique supérieures à celles que nous obtenons avec WattsUp, et ce même si PowerMon2 ne tient pas compte de la consommation du bloc d'alimentation et de certaines consommant internes. Nous avons également souligné que contrairement aux mesures externes, les valeurs obtenues avec les wattmètres internes sont plus étalées et génèrent plus de valeurs aberrantes. Ceci est dû à la fréquence de mesure qui est relativement élevée dans les wattmètres internes.

Contrairement aux profils électriques issus des wattmètres externes, ceux que nous obtenons avec les dispositifs internes sont légèrement différents et présentent des fluctuations électriques particulièrement perceptibles avec NI. Ce comportement peut s'expliquer par la fréquence de mesure très élevée de NI. Par ailleurs, les wattmètres externes d'une part et internes d'autre part fournissent des profils électriques différents qui sont décalés dans le temps durant l'exécution de `hdparm` et plus particulièrement pendant `cpuburn`. Contrairement à Intel_Desktop, le profil électrique sur la machine AMD_Server permet clairement d'observer des pics et creux lorsque nous exécutons le banc d'essai `cpuburn`. Ce comportement électrique est lié aux nombreux ventilateurs dont la vitesse de fonctionnement varie en permanence afin de maintenir la machine à une température constante.

Après avoir tracé les profils électriques en variant la fréquence de mesure, nous avons démontré que mesurer à fréquence très élevée (plus de 500 Hz) n'est pas toujours nécessaire pour étudier le profil électrique d'une application donnée et peut même engendrer des fluctuations électriques que l'on ne peut pas toujours comprendre et interpréter. La fréquence de mesure la plus appropriée n'est donc pas toujours celle qui est la plus élevée, mais celle-ci dépend de l'application. Par ailleurs, un wattmètre interne tel que PowerMon2 permet de fournir des profils temporels de la puissance électrique distincts pour chacune des différentes voies électriques internes : 3.3 V, 5 V et 12 V. Ceci permet notamment de détecter d'où proviennent les fluctuations électriques. On pouvait espérer que chaque composant interne soit alimenté exclusivement par une même ligne interne, mais ce n'était le cas ni pour Intel_Desktop ni pour AMD_Server.

En somme, grâce à une fréquence de mesure élevée et grâce aux différentes voies

électriques mesurées, les dispositifs de mesure de la puissance électrique interne permettent de mieux visualiser certaines fluctuations électriques que les wattmètres externes sont incapables de relever. Toutefois, une fréquence de mesure élevée n'est pas toujours nécessaire pour étudier l'évolution de la puissance électrique durant l'exécution d'un banc d'essai. En outre, la supervision de systèmes distribués à très grande échelle (comme les supercalculateurs exaflopiques) à l'aide des wattmètres internes n'est pas faisable, spécialement parce que ces équipements ne sont pas faciles à brancher et que le prix pour chaque noeud est très onéreux (e.g. 2700 € pour NI).

Pour mesurer la puissance électrique d'un service applicatif, nous préférons utiliser des wattmètres externes plutôt que des wattmètres internes parce que nous n'avons pas besoin de connaître toutes les petites variations de puissance électriques étant donné que celles-ci ne sont pas toujours faciles à comprendre. Nous avons donc plutôt besoin de récupérer une mesure de puissance électrique moyenne qui permet d'inclure quasiment toutes les fluctuations qui ont eu lieu au cours d'une seconde. En plus d'être moins chers et plus faciles à brancher, ces équipements externes prennent en compte la puissance électrique de tous les composants (y compris le bloc d'alimentation et les ventilateurs). Entre OmegaWatt et WattsUp, nous optons pour OmegaWatt car il est plus précis et coûte moins cher. Il est aussi plus commode à brancher pour des plates-formes distribuées étant donné qu'un seul équipement OmegaWatt permet de mesurer les puissances électriques de plusieurs machines simultanément.

3.3 A-t-on besoin de mesurer la puissance électrique pour tous les noeuds d'une même grappe ?

Dans cette troisième section de ce chapitre, nous cherchons à savoir si nous avons besoin de mesurer la puissance électrique pour tous les noeuds d'une même grappe de calcul. Pour répondre à cette question, nous avons besoin de savoir si les noeuds d'une même grappe (identiques sur le plan matériel) ont la même puissance électrique. Nous avons déjà observé dans la section 3.1 qu'il y avaient des différences de puissance électrique lors de l'exécution des opérations liées aux deux services applicatifs étudiés. Dans cette troisième section, nous mettons en évidence à nouveau ces différences lors de l'exécution de quelques bancs d'essai et cherchons à identifier les origines de cette hétérogénéité de la puissance électrique. Pour cette étude, nous utilisons les trois grappes de calcul présentées précédemment dans la section 3.1.1.

3.3.1 Les machines d'une même grappe ont-elles une puissance électrique identique ?

Dans cette section, nous mettons en exergue les différences de puissance électrique entre les noeuds identiques issus d'une grappe homogène (que l'on a déjà pu observer dans la section 3.1). Dans un premier temps, nous exécutons des bancs d'essai spécifiques sur des noeuds provenant d'une grappe homogène donnée. Ensuite,

nous cherchons à montrer que les premières conclusions faites à propos de la grappe de calcul considérée peuvent se généraliser à d'autres grappes de calcul. Enfin, nous expliquons les origines de ces différences en nous appuyant sur l'analyse du comportement électrique de noeuds identiques lorsqu'ils sont au repos : les machines sont allumées mais n'exécutent rien à part le système d'exploitation.

Nous commençons d'abord par nous demander si les noeuds de calculs provenant d'une même grappe avaient les mêmes performance. Pour cela, nous avons mesuré le nombre d'opérations à virgule flottante par seconde [13] caractérisant chaque noeud des trois grappes considérées, mais nous n'avons trouvé que des différences marginales entre les noeuds d'une même grappe. Les noeuds d'une même grappe présentent donc les mêmes performance.

Ensuite, nous avons cherché à montrer que les noeuds de calcul provenant d'une même grappe et exécutant les mêmes bancs d'essai ne consomment pas la même puissance électrique. Afin de répondre à cette question, nous choisissons la grappe *Sagittaire* puisqu'il s'agit de la grappe homogène qui présente le plus grand nombre de noeuds de calcul. Nous exécutons simultanément sur chaque noeud de calcul de la grappe *Sagittaire* différents bancs d'essai qui sollicitent intensément des parties spécifiques des machines et nous mesurons la puissance électrique de chaque noeud durant chaque banc d'essai. Parmi les bancs d'essai présentés dans la section 3.2.1.3, nous en considérons trois en particulier : `cpuburn`, `burnMMX` et `hdparm` qui respectivement sollicitent le processeur, la mémoire RAM et le disque dur. Nous exécutons chaque banc d'essai en même temps sur tous les noeuds de la grappe afin de s'assurer que les conditions environnementales ont été les mêmes durant les mesures de puissance électrique.

Sur chaque noeud, nous exécutons successivement les scénarios suivants :

1. Tous les coeurs de calcul exécutent `cpuburn` pendant 60 secondes ;
2. tous les coeurs de calcul exécutent `burnMMX` pendant 60 secondes ;
3. un seul coeur de calcul exécute `hdparm` pendant 60 secondes.

La figure 3.21 présente le profil temporel de puissance électrique obtenu sur les 60 noeuds de calcul de la grappe *Sagittaire*. Elle montre que les profils électriques des noeuds restent à peu près constants pendant les 60 secondes de `cpuburn` et pendant celles de `burnMMX`. Comme nous l'avons vu dans la section 3.2, nous observons à nouveau les oscillations de la puissance électrique pendant les 60 secondes d'exécution du banc d'essai `hdparm`. Ceci est dû au fait que ce banc d'essai alterne des phrases de lectures et des écritures sur le disque dur qui ne nécessitent pas la même puissance électrique.

De plus, la figure 3.21 montre que les noeuds identiques de la grappe *Sagittaire* ne consomment pas la même puissance électrique bien qu'ils exécutent les mêmes bancs d'essai dans les mêmes conditions environnementales. Pour le banc d'essai `cpuburn`, le noeud le moins consommateur affiche une puissance électrique d'environ 225 W tandis que le plus consommateur nécessite environ 275 W. Cela représente un écart

13. FLOPS : *FLoat Operations Per Second*

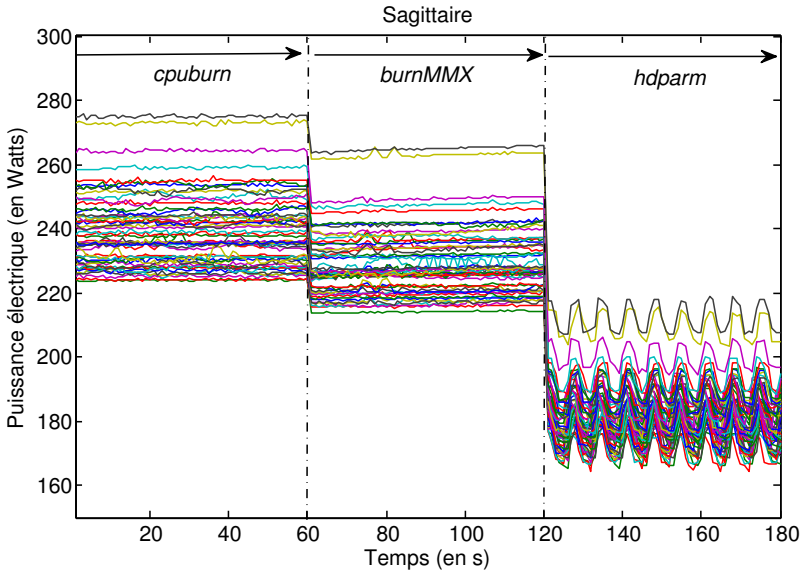

FIGURE 3.21 – Puissance électrique des 60 noeuds identiques de la grappe *Sagittaire*

d'environ 22%. Pour le banc d'essai `burnMMX`, la puissance électrique varie de 215 W à 265 W. Quant à `hdparm`, la puissance électrique varie entre 165 W et 215 W si on se place au niveau des creux et entre 170 W et 220 W si on considère les pics.

À présent, nous cherchons à savoir si nous observons les mêmes différences en termes de puissance électrique pour des noeuds issus d'autres grappes homogènes : *Stremi* et *Taurus*. Pour chaque grappe homogène (*Sagittaire*, *Stremi* et *Taurus*), nous considérons les mêmes scénarios que ceux que nous avons précédemment décrits dans la section 3.1.4 mais pour une durée de 10 minutes afin de relever plus de points de mesure. Étant donné que les profils électriques des bancs d'essai `cpuburn` et `burnMMX` restent constants pour chacun des noeuds et que `hdparm` oscille uniformément autour d'une valeur moyenne, nous avons calculé pour chaque noeud, la puissance électrique moyenne sur les 10 minutes et ce durant chaque scénario.

La figure 3.22 présente les *boîtes à moustaches* montrant la variabilité de la puissance électrique des noeuds de chaque grappe lorsqu'ils exécutent les bancs d'essai `cpuburn`, `burnMMX` et `hdparm`. Les *boîtes à moustaches* représentent les puissances moyennes des 60 noeuds de *Sagittaire*, des 42 noeuds de *Stremi* et des 16 noeuds de *Taurus*. Afin de faciliter la comparaison, les *boîtes à moustaches* sont affichées dans la même échelle.

Premièrement, la figure 3.22 montre que pendant un banc d'essai donné, la puissance électrique n'est pas la même pour des noeuds identiques issus de différentes grappes homogènes. Comme nous l'avons précédemment vu dans la section 3.1.4 pour la grappe *Sagittaire* exécutant `cpuburn`, la puissance électrique oscille approximativement entre 225 W pour le noeud le moins consommateur et 275 W pour le plus consommateur. Cette différence de 50 W est très importante et représente environ un écart de 22 %.

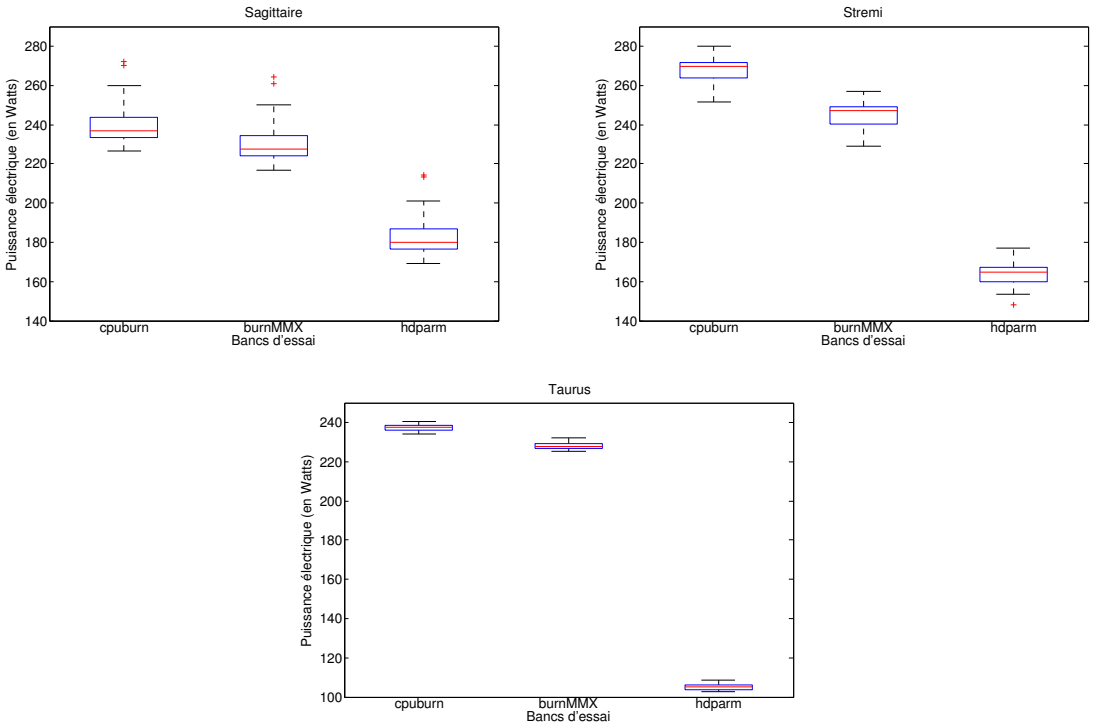

FIGURE 3.22 – Puissance électrique de machines issues de trois différentes grappes homogènes

L'étendue des puissances électriques par noeud est aussi très importante pour la grappe *Stremi* : la puissance électrique moyenne par noeud exécutant `cpuburn` s'échelonne approximativement entre 250 W à 280 W. Cependant, pour la grappe *Taurus*, cette étendue n'est pas aussi importante. En effet, la puissance électrique moyenne par noeud s'étale approximativement entre 235 W et 240 W lorsque tous les coeurs de calcul de chaque noeud exécutent `cpuburn`. Nous constatons les mêmes étendues en ce qui concerne les puissances électriques avec les bancs d'essai `burnMMX` et `hdparm`.

En outre, la figure 3.22 montre que pour un banc d'essai donné, les puissances électriques des machines d'une même grappe de calcul homogène ne sont pas toujours uniformément réparties sur tout l'intervalle de la puissance électrique. En effet, la plupart des noeuds issus d'une même grappe homogène ont une puissance électrique proche de la puissance électrique médiane de tous les noeuds. Pour 50% des noeuds provenant de la grappe *Sagittaire* et exécutant `cpuburn`, la puissance électrique moyenne par noeud s'étale approximativement entre 235 W et 245 W. Quant aux 50% de noeuds restants, la puissance électrique moyenne par noeud lors de l'exécution de `cpuburn` est en dehors de l'intervalle [235W, 245W]. En ce qui concerne la grappe *Stremi*, 50% des noeuds exécutant `cpuburn` sur tous les coeurs de calcul présentent une puissance électrique moyenne par noeud se situant approximativement entre 262 W et 272 W tandis que les 50% des noeuds restants ont des puissances électriques en dehors de cet intervalle. Toutefois, ce n'est pas vraiment le cas pour la grappe *Taurus* dans la mesure où les noeuds semblent être uniformément distribués. Pour une grappe de machines donnée, nous notons des répartitions similaires de la puissance électrique moyenne par noeud avec les bancs d'essai `burnMMX` et `hdparm`.

Les premières observations que nous avons tirées en ce qui concerne l'étendue et la distribution de la puissance électrique des noeuds de la grappe *Taurus* ne sont pas aussi importantes qu'elles le sont pour les grappes *Sagittaire* et *Stremi*. Ceci peut être dû au fait que cette grappe n'est composée que de 16 machines tandis qu'il y a 42 noeuds dans la grappe *Stremi* et 60 noeuds dans *Sagittaire*. Une autre explication peut être liée à l'âge de la grappe *Taurus*. En effet, les noeuds de cette grappe ont été utilisés seulement pendant 9 mois et pour cette raison, ils sont peut-être encore relativement homogènes sur le plan de la puissance électrique. Nous détaillerons ce point dans la section 3.3.2.

Enfin, pour une grappe donnée, nous remarquons que la répartition des puissances électriques est presque la même lorsque nous considérons différents bancs d'essai. En effet, pour une grappe donnée, la répartition des puissances électriques semble subir une translation sur l'axe des ordonnées lorsque nous passons d'un banc d'essai à l'autre. Par exemple, avec la grappe *Sagittaire*, la puissance électrique moyenne par noeud s'étale approximativement de 225 W à 275 W pour `cpuburn`, de 215 W à 265 W pour `burnMMX` et de 165 W à 215 W pour `hdparm`. Ainsi, pour la grappe *Sagittaire*, nous observons toujours une différence de 50 W quel que soit le banc d'essai considéré. De la même manière, l'étendue est de 30 W pour *Stremi* et de 10 W pour *Taurus*. Ces dernières observations nous amènent à nous demander

si cette hétérogénéité de la puissance électrique demeure perceptible même lorsque les noeuds sont au repos, c'est-à-dire allumés mais n'exécutant rien à l'exception du système d'exploitation.

À présent, nous analysons la puissance électrique au repos des noeuds issus des trois différents grappes homogènes. Dans cette optique, nous mesurons pendant 10 minutes la puissance électrique de chaque noeud lorsqu'il est au repos, c'est-à-dire allumé et n'exécutant rien à part le système d'exploitation.

Pour chaque grappe homogène, nous calculons la puissance électrique moyenne par noeud durant ces 10 minutes d'inactivité. Nous remarquons sur la figure 3.23 que la puissance électrique au repos n'est pas la même pour des noeuds identiques issus d'une même grappe. En effet, la puissance électrique au repos de la grappe *Sagittaire* s'échelonne approximativement entre 165 W pour le noeud le moins consommateur et 215 W pour le plus consommateur. La puissance électrique au repos de la grappe *Stremi* s'échelonne approximativement entre 140 W et 170 W, celle de *Taurus* va de 95 W à 100W environ. Pour chaque grappe homogène, l'étendue entre les deux noeuds extrêmes en termes de puissance électrique est la même que celle que nous avons observée pour les noeuds en activité exécutant un banc d'essai spécifique : 50 W pour *Sagittaire*, 30 W pour *Stremi* et 5 W pour *Taurus*.

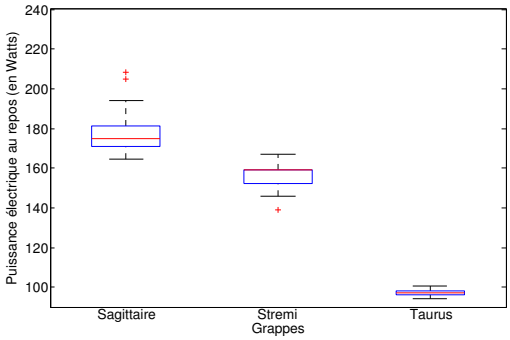

FIGURE 3.23 – Puissance électrique au repos des noeuds issus de trois différentes grappes homogènes

Pour chaque grappe homogène, les *boîtes à moustaches* pour les noeuds au repos semblent être superposables à celles que nous avons observées pour les noeuds exécutant un banc d'essai donné. Cela signifie que la distribution des puissances électriques reste identique même avec des noeuds au repos. Par conséquent, l'hétérogénéité des noeuds identiques en termes de puissance électrique n'est pas liée à la nature du programme qu'ils exécutent.

Maintenant que nous savons que des noeuds identiques ne consomment pas la même puissance électrique lorsqu'ils exécutent un banc d'essai ou lorsqu'ils sont au repos, nous souhaiterions savoir si les différences de consommation que nous avons observées lorsque les noeuds exécutent un banc d'essai sont exclusivement dues aux différences en termes de puissance électrique au repos. Ainsi, pour chaque banc

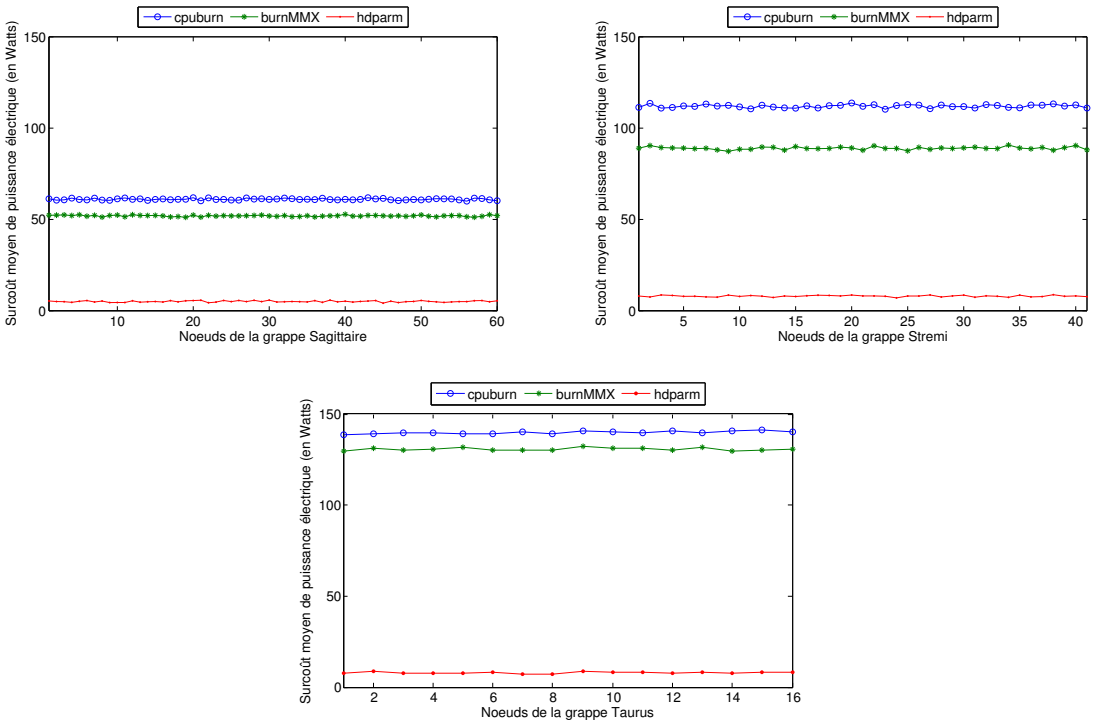

FIGURE 3.24 – Surcoût de puissance électrique des noeuds provenant de trois différentes grappes homogènes et exécutant différents programmes

d'essai, nous soustrayons la puissance électrique au repos à la puissance électrique consommée pendant l'exécution d'un banc d'essai donné. Nous avons représenté sur la figure 3.24 ce surcoût de puissance électrique lié à chaque banc d'essai pour les trois grappes considérées.

La figure 3.24 montre que le surcoût de puissance électrique moyenne par noeud pendant un banc d'essai spécifique est le même pour tous les noeuds d'une même grappe homogène. En effet, pour la grappe *Sagittaire*, ce surcoût de puissance électrique par noeud est constant et vaut environ 62 W pour `cpuburn`, de 52 W pour `burnMMX` et de 5 W pour le banc d'essai `hdparm`. Cela montre que les différences de puissance électrique par noeud pendant un programme donné sont presque entièrement dues aux différences en termes de puissance électrique au repos. Nous en déduisons que nous avons besoin de mesurer la puissance électrique au repos de chacun des noeuds d'une grappe de calcul. Par contre, nous n'avons besoin de mesurer le surcoût de puissance électrique liée à une opération que pour un seul noeud de la grappe. Dans la section suivante, nous nous intéressons à analyser pourquoi nous observons ces différences en termes de puissance électrique au repos alors que les noeuds sont identiques d'un point de vue matériel.

3.3.2 Quelle est l'origine des différences de puissance électrique au repos ?

Dans cette section, nous cherchons à trouver une explication aux différences de puissance électrique au repos. La section 3.3.2.1 présente la méthodologie expérimentale et les sections suivantes détaillent nos investigations successives.

3.3.2.1 Méthodologie expérimentale

Afin de trouver l'origine de telles différences, nous avons extrait de la grappe *Sagittaire* deux noeuds dont la puissance électrique est très différente : *sagittaire-54* et *sagittaire-57* qui respectivement consomment 167 W et 205 W lorsqu'ils sont au repos (une différence de 38 W).

D'abord, nous avons soupçonné que les conditions de mesure de la puissance électrique n'étaient pas les mêmes pour ces deux noeuds. Nous nous sommes demandés si ces différences pouvaient être liées aux conditions environnementales, par exemple à la position des noeuds dans l'armoire des serveurs, ou à un défaut d'étalonnage de nos wattmètres. Ensuite, nous avons exploré en détails la puissance électrique interne des différents composants de chacun des deux noeuds. Nous avons permuté quelques composants internes (blocs d'alimentation, disques durs, et barrettes de mémoire) des deux noeuds considérés dans l'optique de découvrir ceux qui étaient hétérogènes en termes de puissance électrique. Enfin, nous avons mesuré la puissance électrique des processeurs et des ventilateurs les refroidissant en utilisant PowerMon2 [Bedard et al., 2010].

La figure 3.25 présente une photographie du dispositif expérimental.

FIGURE 3.25 – Noeud *sagittaire-54* sur lequel est branché POWERMON2

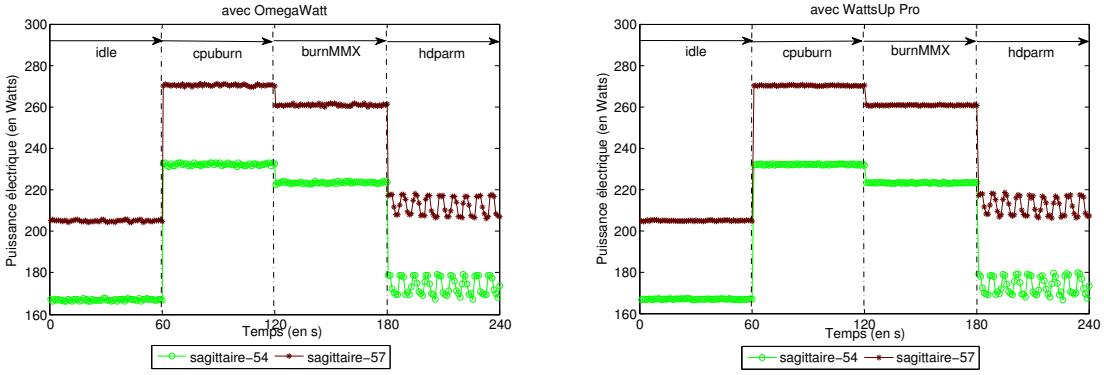

FIGURE 3.26 – Comparaison des profils électriques de *sagittaire-54* et *sagittaire-57* obtenues à l'aide de OMEGAWATT (à gauche) et WATTSUP (à droite)

3.3.2.2 Conditions environnementales

Nous avons d'abord pensé que même si les conditions expérimentales étaient les mêmes pendant nos mesures de puissance électrique, la température pouvait ne pas être uniformément répartie dans la pièce où nous avons effectué nos mesures, et que la position des noeuds dans l'armoire des serveurs pouvait expliquer les écarts de puissance électrique de deux noeuds identiques au repos, comme cela a été mentionné dans les articles [Varsamopoulos et al., 2009, Tang et al., 2007]. Nous avons donc permuté les positions de ces deux noeuds et mesuré la puissance électrique dans ces nouvelles positions. Les puissances électriques au repos des deux noeuds n'ont pas changé et *sagittaire-54* est toujours le noeud qui consomme le moins. Dans notre cas, la position dans l'armoire n'est donc pas la cause des différences de puissance électrique que nous avons observées.

Notre deuxième hypothèse était que les wattmètres utilisés n'étaient pas étalonnés et que les mesures obtenues sur chaque prise étaient biaisées par rapport aux consommations réelles. Pour vérifier cela, nous avons utilisé l'autre wattmètre externe dont nous disposons : WattsUp. La figure 3.26 présente la puissance électrique mesurée par les deux différents wattmètres durant différents bancs d'essai. Elle montre que les puissances électriques affichées par WattsUp sont sensiblement les mêmes que celles que nous avons obtenues avec OmegaWatt. Ainsi, ces différences en termes de puissance électrique au repos ne sont pas dues aux wattmètres que nous utilisons.

3.3.2.3 Composants matériels spécifiques

Ensuite, nous avons supposé que ces différences pouvaient provenir des blocs d'alimentation des noeuds qui auraient pu perdre leur efficacité avec le temps et générer ainsi quelques fluctuations électriques. Pour cela, nous avons échangé les blocs d'alimentation électrique de ces deux noeuds et nous avons mesuré la puissance électrique au repos. Cette dernière est restée la même pour *sagittaire-54* et pour *sagittaire-57*. Les blocs d'alimentation électrique ne sont donc pas en cause.

Nous avons permuté d'autres composants matériels internes de *sagittaire-54* et de *sagittaire-57* : les disques durs et les barrettes de cartes mémoire. Les mesures de la puissance électrique au repos pour ces deux noeuds sont restées inchangées après avoir permuté leurs disques durs et leurs barrettes de cartes mémoire. Ainsi, les différences de puissance électrique au repos ne sont dues ni aux disques durs ni aux barrettes de cartes mémoire.

3.3.2.4 Analyse de la puissance électrique des processeurs

Nous aurions souhaité échanger les processeurs des deux noeuds mais nous n'avons pas pu car les processeurs sont collés à la carte mère et leur permutation risquait d'endommager les noeuds. Au lieu de cela, nous avons utilisé PowerMon2 afin de mesurer indépendamment les lignes de 12 V qui alimentent les processeurs et les ventilateurs associés pour les noeuds *sagittaire-54* et *sagittaire-57*. La figure 3.27 montre les profils de puissance électrique établis par les bancs d'essai `idle`, `cpuburn`,

`burnMMX` et `hdparm`. Il est important de noter que la puissance électrique interne affichée sur la figure 3.27 inclut seulement celle des processeurs et des ventilateurs associés. Ceci explique les différences par rapport aux mesures affichées sur la figure 3.26.

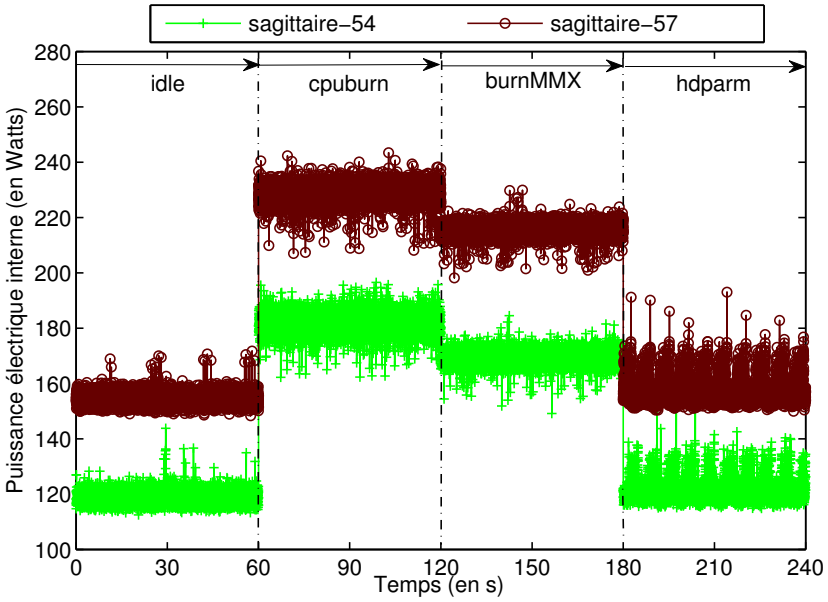

FIGURE 3.27 – Puissance électrique des processeurs et des ventilateurs associés des deux noeuds spécifiques de la grappe *Sagittaire* pendant l'exécution de différents bancs d'essai

D'abord, nous remarquons sur la figure 3.27 que lorsque les deux noeuds sont au repos, la puissance électrique des processeurs et des ventilateurs associés est approximativement de 118 W pour *sagittaire-54* et de 155 W pour *sagittaire-57*, soit un écart de 37 W. Nous retrouvons presque la différence de 38 W que nous avons observée à l'aide de OMEGAWATT. Ainsi, nous en concluons que la différence de puissance électrique est due à la différence de puissance électrique au repos des processeurs et des ventilateurs associés.

À partir de cette puissance électrique interne, nous observons que la consommation des processeurs et des ventilateurs associés représente une grande portion de la puissance électrique totale du noeud, et ce même lorsqu'il est au repos. En effet, elle est de 118 W (figure 3.27) contre 167 W (figure 3.26) pour tout le noeud *sagittaire-54* et de 155 W contre 205 W pour tout le noeud *sagittaire-57*. Cela représente un ratio de 70% pour *sagittaire-54* et de 75% pour *sagittaire-57*.

Considérons que la puissance électrique des processeurs et des ventilateurs associés suit le modèle suivant :

$$P_{CPUs\&ventilateurs} = P_{repos}^{statique} + \Delta P_{application}^{dynamique} \tag{3.2}$$

$P_{repos}^{statique}$ est la puissance électrique des processeurs et des ventilateurs associés lorsque le noeud est au repos. $\Delta P_{application}^{dynamique}$ est le surcoût de puissance électrique des

processeurs et des ventilateurs associés lors de l'exécution d'une application sur le noeud en question.

Même avec différents bancs d'essai (`cpuburn`, `burnMMX` et `hdparm`), nous observons encore la différence de 38 W, ce qui confirme que cette différence est constante quel que soit le programme exécuté par les deux machines. Ainsi, cette différence de puissance électrique n'est pas due à la part dynamique liée au banc d'essai et représentée par $\Delta P^{dynamique}_{application}$ mais à la différence en termes de puissance électrique au repos $P^{statique}_{repos}$ des processeurs et des ventilateurs associés.

Les noeuds de la grappe *Sagittaire* ayant les mêmes performances (mêmes *FLOPS*), il semble donc qu'il n'y a pas de relation de causalité entre l'hétérogénéité de puissance électrique des processeurs et leur performance de calcul.

En raison des fluctuations dans les processus de fabrication, différents processeurs peuvent consommer différemment, même s'ils ont les mêmes caractéristiques techniques et fonctionnent à la même fréquence. En effet, les noeuds des grappes *Sagittaire* et *Stremi* sont constitués de processeurs AMD contrairement aux noeuds de la grappe *Taurus* qui ont des processeurs Intel. Les fluctuations dans les processus de fabrication ont peut-être un impact plus conséquent sur les processeurs AMD. De plus, pour les grappes *Sagittaire* et *Stremi*, une explication complémentaire est qu'après certaines défaillances, les processeurs et/ou les ventilateurs chargés de les refroidir peuvent être usés, engendrant ainsi des puissances électriques hétérogènes. Ceci peut aussi être dû à une accumulation hétérogène de la poussière. Une dernière hypothèse serait qu'une part croissante de la puissance électrique provient des courants de fuite qui peuvent varier au cours du temps en raison des processus d'électromigration [Black, 1969]. Ceci pourrait expliquer le fait que nous observons des différences significatives en termes de puissance électrique au repos pour la grappe *Sagittaire* qui a fonctionné pendant 8 années en comparaison à la grappe *Taurus* qui n'a fonctionné que pendant 9 mois. En effet, les différences observées dans la grappe *Taurus* sont marginales (environ 5%) et peuvent être dues à l'hétérogénéité dans les processus de fabrication. Donc, même si l'utilisateur final peut avoir l'impression de bénéficier d'une grappe complètement homogène, l'hétérogénéité existe au moins sur le plan électrique.

3.4 Conclusions du chapitre

Dans ce chapitre, nous avons commencé par chercher à mesurer la puissance électrique des opérations que nous retrouvons dans les services applicatifs de la tolérance aux pannes et de la diffusion de données. **Nous avons exposé notre méthodologie pour mesurer la puissance électrique lors de ces opérations.**

Étant donné que les wattmètres externes utilisés fournissent au mieux une mesure par seconde, nous avons besoin que le temps d'exécution de l'opération du service applicatif dont nous souhaitons mesurer la puissance électrique, soit au moins supérieure à une seconde. Pour avoir plusieurs points de mesure, il est nécessaire que ce

temps d'exécution soit de quelques secondes. Pour ce faire, nous essayons dans un premier temps de faire varier les paramètres d'exécution afin que l'opération pour laquelle nous cherchons à mesurer la puissance électrique dure quelques secondes. Cela a été possible pour la plupart des opérations. Quand cela n'a pas été possible, nous avons instrumenté l'implémentation du service applicatif afin de répéter spécifiquement l'opération pour laquelle nous voulons mesurer la puissance électrique. Cela a été nécessaire uniquement pour la synchronisation et pour l'enregistrement de messages. Par ailleurs, nos mesures de la puissance électrique ont pu montrer que **le nombre de processus déployés par noeud peut faire varier la puissance électrique consommée par le noeud**. Ceci a été le cas lorsque nous avons fait varier le nombre de processus par noeud qui sont impliqués dans la diffusion de données (section 3.1.3) ou lorsqu'un nombre variable de coeurs par noeud exécutaient `cpuburn` (figure 3.10).

Nous avons été confrontés notamment à deux difficultés. D'une part, il est important de choisir un équipement de mesure suffisamment précis pour que les mesures relevées soient fiables. D'autre part, même s'ils sont identiques, des noeuds d'une même grappe de calcul ne consomment pas forcément la même puissance électrique bien qu'ils exécutent exactement les mêmes opérations.

Pour choisir le bon équipement de mesure, nous avons étudié et comparé différents dispositifs de mesures externes et internes mesurant la puissance électrique à des fréquences plus ou moins importantes. Nous avons d'abord démontré qu'il n'est pas facile d'obtenir des mesures électriques très précises même avec des équipements de mesure affichant une précision élevée dans leurs caractéristiques techniques. Nous avons montré qu'une fréquence de mesure élevée n'est pas toujours nécessaire pour étudier l'évolution de la puissance électrique durant l'exécution d'une application donnée.

Dans notre cas, **nous optons pour le wattmètre externe** OmegaWatt car il est capable de fournir une mesure de puissance électrique moyenne qui comprend toutes (ou quasiment toutes) les fluctuations qui ont eu lieu au cours d'une seconde. En plus d'être moins chers et plus faciles à brancher, les équipements externes prennent en compte la puissance électrique de tous les composants (y compris le bloc d'alimentation et les ventilateurs). De plus, nous optons pour OmegaWatt plutôt que WattsUp dans la mesure où il est plus précis et est aussi plus commode à brancher pour des plates-formes distribuées étant donné qu'un seul équipement OmegaWatt permet de mesurer les puissances électriques de plusieurs machines simultanément.

Enfin, l'étude de la puissance électrique de noeuds issus d'une même grappe de calcul a permis de montrer que même s'ils sont constitués des mêmes composants matériels, des noeuds identiques peuvent avoir une puissance électrique hétérogène. Cette étude nous a montré deux résultats importants. D'une part, **l'hétérogénéité électrique des noeuds identiques d'une même grappe est due principalement aux différences de la puissance électrique de ces noeuds lorsqu'ils sont au repos**. D'autre part, **le surcoût dynamique de la puissance élec-**

trique qui est dû à l'exécution d'une d'application (ou une partie) est homogène pour des noeuds identiques issues d'une même grappe. Une méthodologie expérimentale nous a permis de détecter que cette hétérogénéité de la puissance électrique au repos provenait principalement des processeurs et des ventilateurs chargés de les refroidir. Cette méthodologie nous a également permis de constater que les performances des noeuds ne sont pas affectées par cette hétérogénéité électrique, étant donné que ces noeuds ont **les mêmes performances (*i.e.*, mêmes *FLOPS*)**.

4 Estimation de la consommation énergétique des services applicatifs

Le chapitre précédent nous a appris qu'il est difficile de mesurer précisément la consommation énergétique d'une application ou d'un service applicatif s'exécutant sur une plate-forme distribuée à très large échelle. Bien que certains dispositifs puissent y surmonter cette difficulté, la mesure de la consommation énergétique nécessite toujours d'exécuter l'application ou le service applicatif à très grande échelle et ceci dans tous les contextes d'exécution. Afin de réduire le nombre de mesures, nous devons être capable d'estimer de façon précise la consommation énergétique du service applicatif, et ce pour n'importe quel contexte d'exécution et pour n'importe quelle plate-forme expérimentale. L'intérêt de cette estimation est d'évaluer la consommation énergétique d'un service applicatif **sans avoir à le pré-exécuter dans chaque contexte d'exécution**, et ce dans l'optique d'être en mesure de choisir la version de ce service qui consomme le moins d'énergie.

Afin que les estimations énergétiques soient adaptées à la plate-forme d'exécution, nous avons besoin de collecter un ensemble de mesures de la puissance électrique des noeuds de la plate-forme lors des différentes opérations qui composent un service applicatif. Or, le chapitre précédent nous a appris que les noeuds d'une même grappe peuvent avoir une puissance électrique au repos hétérogène mais qu'ils ont le même surcoût dynamique de la puissance électrique lié à l'exécution d'une d'application (ou une partie). Nous en déduisons que **nous avons besoin de mesurer la puissance électrique au repos de chacun des noeuds d'une grappe de calcul**

mais que nous n'avons besoin de mesurer le surcoût de puissance électrique liée à une opération que pour chaque type de noeuds. Par ailleurs, pour estimer la consommation énergétique en fonction de la plate-forme d'exécution, nous avons également besoin de mesurer le temps d'exécution de chacune des opérations sur cette plate-forme. Or, le chapitre précédent nous a appris que les noeuds d'une même grappe étaient homogènes en termes de performances. Nous en déduisons que **nous n'avons besoin de mesurer le temps d'exécution lié à une opération que pour chaque type de noeuds**. Nous appelons calibration le processus qui consiste à orchestrer la prise de ces mesures de puissance électrique et de temps d'exécution. Afin que nos estimations énergétiques soient conformes au contexte d'exécution, notre approche d'estimation repose également sur une description des paramètres d'exécution fournie par l'utilisateur.

Dans ce chapitre, nous expliquons notre méthodologie d'estimation depuis l'identification des opérations du service applicatif jusqu'aux modèles d'estimation énergétique de ces opérations, en passant par une description de la calibration et des paramètres d'exécution dont nous avons besoin. Nous appliquons chaque étape de notre méthodologie aux protocoles de tolérance aux pannes [Diouri et al., 2013e, Diouri et al., 2013h, Diouri et al., 2013f] et aux algorithmes de diffusion de données [Diouri et al., 2013g] que nous avons considérés dans le chapitre précédent. Dans la section 4.1, nous identifions les différentes opérations pour les deux services applicatifs étudiés. La section 4.2 présente notre méthodologie pour calibrer la puissance électrique et le temps d'exécution des opérations identifiées. La section 4.3 montre comment nous estimons la consommation énergétique des différentes opérations en nous appuyant sur la calibration et les différents paramètres d'exécution. Dans la section 4.4, nous évaluons la précision des estimations pour chacun des deux services applicatifs en les confrontant aux mesures énergétiques réelles. La section 4.5 présente les conclusions du chapitre.

4.1 Identification des opérations

La première étape de notre méthodologie consiste à identifier les différentes opérations que nous retrouvons dans les différentes versions d'un service applicatif. Une opération est une tâche que le service applicatif peut devoir effectuer plusieurs fois pendant l'exécution d'une application. Dans les sections 4.1.1 et 4.1.2, nous identifions les différentes opérations des services de la tolérance aux pannes et de la diffusion de données. Dans la section 4.2, nous présenterons les différents paramètres dont dépend la consommation énergétique des opérations liées à la tolérance aux pannes et à la diffusion de données.

4.1.1 Cas de la tolérance aux pannes

Comme nous l'avons décrit dans le chapitre 1 et également dans la section 3.1.2, nous étudions les deux grandes familles de protocoles de tolérance aux pannes : les

protocoles coordonnés et les protocoles non coordonnés. Pour chacune de ces deux familles, nous distinguons deux grandes phases : d'une part, la sauvegarde de points de reprise qui a lieu pendant le fonctionnement normal (*i.e.,* sans pannes) d'une application ; et d'autre part, la récupération de points de reprise qui a lieu à chaque fois qu'une panne se produit. Dans notre étude, nous nous sommes focalisés sur la phase de sauvegarde de points de reprise.

Nous considérons une application utilisant le protocole de tolérance aux pannes qui est exécutée sur N noeuds avec p processus par noeud et que le nombre de processus par noeud est identique pour les N noeuds. Dans les protocoles de tolérance aux pannes, nous identifions les opérations suivantes :

- Sauvegarde de points de reprise : réalisée par les protocoles coordonnés et les protocoles non coordonnés, cette opération consiste à sauvegarder une image instantanée de l'état courant de l'application. Cette sauvegarde peut être utilisée plus tard pour redémarrer l'exécution de l'application en cas d'une défaillance. Dans notre étude, nous considérons la sauvegarde de points de reprise au niveau système et non pas au niveau applicatif. Un tel choix se justifie par le fait que toutes les applications n'intègrent pas une sauvegarde globale de points de reprise. De plus, la sauvegarde au niveau applicatif ne permet pas de choisir l'intervalle de temps optimal entre deux points de reprise successifs. Nous avons utilisé la sauvegarde de points de reprise fournie par la librairie BLCR (*Berkeley Lab Checkpoint/Restart*) qui est disponible dans l'implémentation MPICH2. Dans la sauvegarde de points de reprise, l'opération de base est l'écriture du point de reprise de taille V_{data} sur un support de stockage donné. Dans notre étude, nous considérons que les points de reprise sont sauvegardés sur le disque dur local de chaque noeud.
- Enregistrement de messages : réalisée dans les protocoles non coordonnés, cette opération consiste à sauvegarder sur un support de stockage donné (mémoire RAM, disque dur local, serveur de stockage distant, etc.) les messages envoyés par chaque processus. En cas de panne, l'enregistrement des messages permet de redémarrer seulement les processus défaillants. Pour l'enregistrement de messages, l'opération de base est l'écriture d'un message de taille V_{data} sur un support de stockage donné. Dans notre étude, nous considérons la mémoire RAM et le disque dur.
- Coordination : réalisée dans les protocoles coordonnés, cette opération consiste à synchroniser les processus juste avant de procéder à la sauvegarde des points de reprise. Si certains processus ont encore des messages en cours de transmission au moment de la coordination, tous les autres processus restent en attente active jusqu'à ce que les messages soient transmis et reçus. Ceci permet de s'assurer qu'il n'y aura pas de messages orphelins : messages qui ont été reçus avant la sauvegarde de points de reprise mais qui n'ont pas été envoyés. Quand il ne reste plus de messages en cours de transmission, tous les processus échangent un message de synchronisation. Dans la coordination, nous pouvons identifier deux sous-opérations qui sont l'attente active durant l'envoi

des messages de taille V_{data} en cours de transmission et la synchronisation de tous les $N \times p$ processus qui a lieu lorsqu'il n'y a plus de message en cours de transmission.

4.1.2 Cas de la diffusion de données

Nous étudions les quatre algorithmes de diffusion de données que nous avons décrits dans la section 3.1.3 à savoir les deux algorithmes utilisés dans MPI *MPI/SAG* et *MPI/Pipeline* et les deux algorithmes hybrides (MPI + OpenMP) *Hybrid/SAG* et *Hybrid/Pipeline*.

Nous rappelons que dans les deux algorithmes hybrides, le processus racine de diffusion de données utilise MPI pour diffuser la donnée à un processus MPI maître par noeud. Ensuite, chaque processus MPI maître utilise OpenMP pour partager la donnée diffusée avec tous les autres processus qui se trouvent dans le même noeud : la routine utilisée dans OpenMP est *CopyPrivate*.

Nous considérons une diffusion de données de taille V_{data} entre N noeuds avec p processus par noeud. Nous supposons que le nombre p de processus par noeud est identique pour les N noeuds. La figure 4.1 présente les quatre algorithmes de diffusion de données que nous considérons et montre les tailles des messages échangés entre les différents processus.

Dans ces quatre algorithmes, nous identifions les opérations suivantes :

- *Scatter* : cette opération consiste à diviser la donnée en plusieurs petits fragments dont le nombre est égal au nombre total de processus et à envoyer chacun des fragments à chaque processus correspondant en utilisant une topologie en arbre binomial. Cette opération est utilisée dans *MPI/SAG* et *Hybrid/SAG*.
- *AllGather* : cette opération consiste à effectuer un échange total des fragments de données envoyés lors du *Scatter* en utilisant une topologie réseau en anneau. À la fin de cette opération, tous les processus auront la totalité de la donnée diffusée. Cette opération est utilisée dans *MPI/SAG* et *Hybrid/SAG*.
- *Pipeline* : cette opération consiste à découper un message initial en un nombre arbitraire de paquets (appelés *chunks*) qui sont acheminés en pipeline. Sur la figure 4.1, ce nombre de paquets est noté C. Cette opération est utilisée dans *MPI/Pipeline* et *Hybrid/Pipeline*.
- *CopyPrivate* : cette opération consiste à copier une donnée stockée dans une variable d'un processus vers les variables correspondantes dans les autres processus qui se trouvent sur le même noeud. Cette opération est utilisée dans *Hybrid/SAG* et *Hybrid/Pipeline*.

4.2 Méthodologie de calibration énergétique

Pour estimer la consommation énergétique de ces opérations, il faut prendre en compte un grand nombre de paramètres :

- paramètres liés au service ou à l'application : intervalle de temps entre la

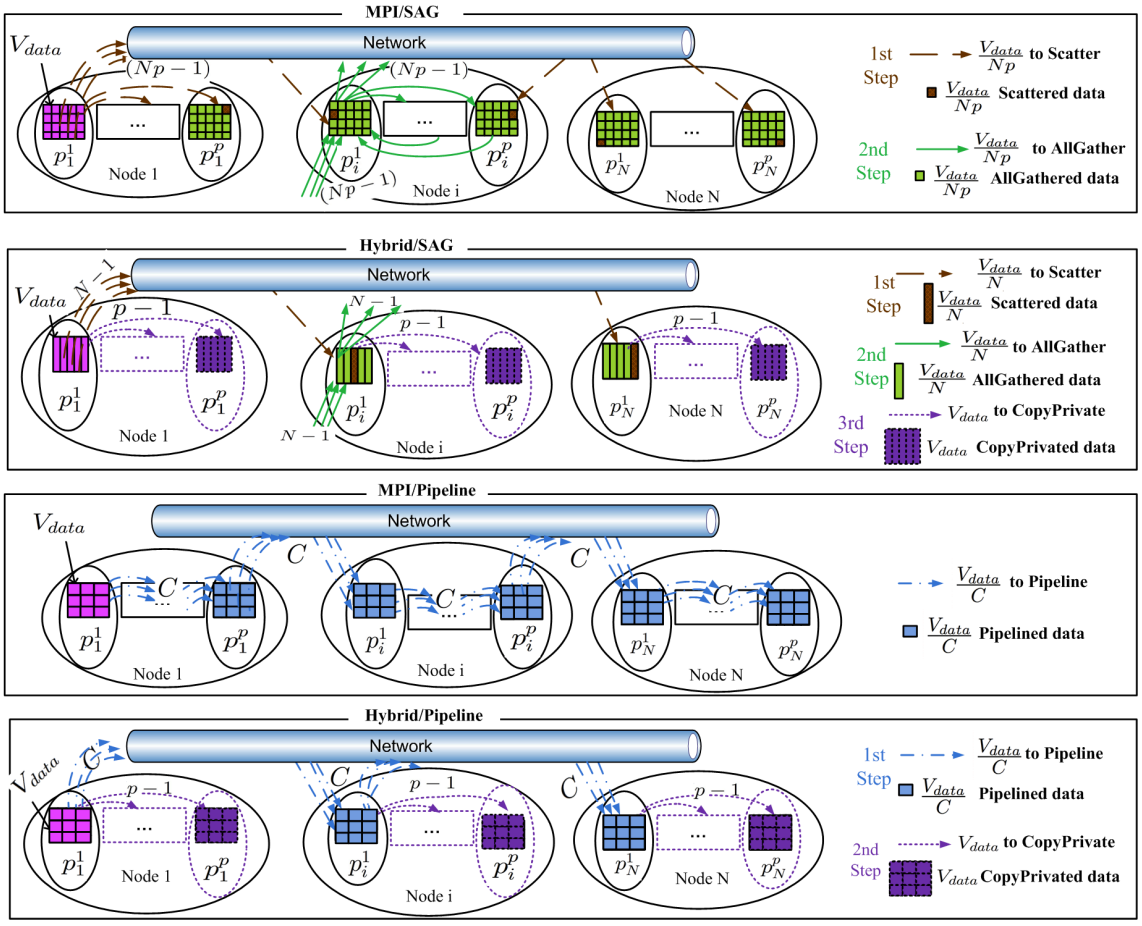

FIGURE 4.1 – Algorithmes de diffusion des données et opérations associées

sauvegarde de deux points de reprise, volume de données à diffuser, nombre de processus, nombre et taille des messages échangées par chaque processus, type de support utilisé pour le stockage (mémoire, disque dur local, disque dur distant, etc.), volume des données écrites ou lues par chaque processus, etc.
- paramètres liés à la plate-forme d'exécution : nombre de noeuds, nombre de processeurs par noeud, nombre de coeurs par processeur, système d'exploitation, topologie réseau, architecture mémoire, technologies réseau (Infiniband, Gigabit Ethernet, solutions propriétaires, etc), type de disques durs (SSD, SATA, SCSI, etc), etc.

Nous considérons qu'un paramètre est une variable de l'estimateur seulement si une variation de ce paramètre peut faire varier considérablement la consommation énergétique pendant que tous les autres paramètres restent fixés. Il est nécessaire de procéder à une calibration de la plate-forme en prenant en compte tous les paramètres pour estimer la consommation énergétique.

La consommation énergétique dépend fortement du matériel utilisé par la plate-forme d'exécution. Par exemple, la consommation énergétique de la sauvegarde de points de reprise dépend du type de support de stockage utilisé (SSD, SATA, SCSI, etc.), de la vitesse de lecture et d'écriture et du temps d'accès à la ressource. Le processus de calibration consiste à collecter une base de connaissances sur l'énergie consommée par les différentes opérations identifiées, et ce en tenant compte du matériel utilisé dans le supercalculateur. Nous récupérons les informations énergétiques sur les opérations identifiées en exécutant un ensemble de bancs d'essai qui permettent de collecter un ensemble de mesures de la puissance électrique et du temps d'exécution des différentes opérations. Le but de cette calibration énergétique est d'adapter les évaluations issues des modèles théoriques d'estimation en fonction du matériel utilisé, et ce dans une perspective de rendre précises nos estimations énergétiques sur n'importe quel supercalculateur, quelles que soient ses caractéristiques. Bien que cette base de connaissances puisse avoir une taille considérable, il est à noter que nous n'avons besoin de procéder à une calibration énergétique que très occasionnellement, par exemple quand il y a un changement de matériel (par exemple, en raison du changement d'un disque dur).

Pour estimer la consommation énergétique d'un noeud effectuant une opération *op*, nous avons besoin de connaître la puissance électrique du noeud lors de l'exécution de *op* ainsi que le temps d'exécution cette opération. Le chapitre précédent nous a appris que les noeuds d'une même grappe étaient homogènes en termes de performances. Nous en déduisons que nous n'avons besoin de mesurer et d'estimer le temps d'exécution liée à une opération que pour chaque type de noeuds. Ainsi, l'énergie $\xi_{op}^{Noeud_i}$ consommée par un noeud i réalisant une opération *op* est :

$$\xi_{op}^{Noeud_i} = \rho_{op}^{Noeud_i} \cdot t_{op}$$

De façon analogue, la consommation énergétique $\xi_{op}^{Switch_j}$ d'un commutateur réseau (*switch*) j pendant l'opération *op* est :

$$\xi_{op}^{Switch_j} = \rho_{op}^{Switch_j} \cdot t_{op}$$

t_{op} est le temps requis pour réaliser l'opération *op* par un type de noeuds.
$\rho_{op}^{Noeud_i}$ est la puissance électrique du noeud i pendant t_{op}.
$\rho_{op}^{Switch_j}$ est la puissance électrique du commutateur réseau j pendant t_{op}.

Par conséquent, pour calibrer la consommation énergétique, nous avons besoin d'un calibrateur de la puissance électrique décrit dans la section 4.2.1 et d'un calibrateur du temps d'exécution décrit dans la section 4.2.2.

4.2.1 Calibration de la puissance électrique ρ_{op}

Nous avons vu dans le chapitre précédent que la puissance électrique d'un noeud i exécutant une opération *op* se décomposait en une partie statique, $\rho_{repos}^{Noeud_i}$, qui est la puissance électrique du noeud i lorsqu'il est au repos et d'une partie dynamique $\Delta\rho_{op}^{Noeud_i}$ qui est le surcoût de puissance électrique lié à l'opération *op* . Dans le chapitre précédent, nous avons montré que $\rho_{repos}^{Noeud_i}$ peut être différente même pour des noeuds identiques issus de grappes homogènes. Par conséquent, nous relevons la mesure de $\rho_{repos}^{Noeud_i}$ pour chaque noeud i. Le chapitre précédent nous a aussi permis de montrer que $\Delta\rho_{op}^{Noeud_i}$ est le même pour des noeuds identiques exécutant la même opération *op*. En effet, nous avons vu que la partie dynamique de la puissance électrique $\Delta\rho_{op}^{Noeud_i}$ ne présentait quasiment pas d'hétérogénéité électrique pour des noeuds issus d'une même grappe homogène. En conséquence, une seule fois pour chaque type de noeuds, nous mesurons $\Delta\rho_{op}^{Noeud_i}$, et ce pour chaque opération *op*.

Dans le chapitre précédent, nous avons aussi vu que le nombre p de processus utilisés par noeud pouvait faire varier la puissance électrique consommée par le noeud. Nous devons donc mesurer $\rho_{op}^{Noeud_i}(p)$ pour chaque opération *op* et pour différentes valeurs de p. Ainsi, la puissance électrique $\rho_{op}^{Noeud_i}(p)$ d'un noeud i lors d'une opération *op* utilisant p processus de ce noeud :

$$\rho_{op}^{Noeud_i}(p) = \rho_{repos}^{Noeud_i} + \Delta\rho_{op}^{Noeud}(p)$$

De façon analogue, la puissance électrique $\rho_{op}^{Switch_j}$ d'un commutateur réseau j pendant l'opération *op* est :

$$\rho_{op}^{Switch_j} = \rho_{repos}^{Switch_j} + \Delta\rho_{op}^{Switch}$$

$\rho_{repos}^{Noeud_i}$ (ou $\rho_{repos}^{Switch_j}$) est la puissance électrique d'un noeud i (ou d'un commutateur réseau j) lorsqu'il est au repos (*i.e.,* allumé mais n'exécutant rien à part le système d'exploitation) et $\Delta\rho_{op}^{Noeud_i}$ (ou $\Delta\rho_{op}^{Switch_j}$) est le surcoût de puissance électrique dû à l'exécution de l'opération *op*.

Pour calculer $\Delta\rho_{op}^{Noeud}(p)$ (ou $\Delta\rho_{op}^{Switch}$), nous mesurons $\rho_{op}^{Noeud_i}(p)$ (ou $\rho_{op}^{Switch_j}$) pour un noeud donné i (ou pour un switch j) et nous soustrayons la partie statique de la puissance électrique qui correspond à la puissance électrique au repos du noeud i (ou du switch j). Nous mesurons $\rho_{op}^{Noeud_i}(p)$ (ou $\rho_{op}^{Switch_j}$) de la même manière que présentée dans le chapitre précédent, dans la section 3.1.1 (c'est-à-dire en faisant durer l'opération quelques secondes). Il s'agit donc d'un surcoût *moyen* de puissance

électrique puisqu'il est calculé à partir de la moyenne de plusieurs mesures (une chaque seconde).

Or, $\rho_{op}^{Noeud_i}(p)$ et donc $\Delta\rho_{op}^{Noeud}(p)$ peuvent varier en fonction du nombre p de processus utilisés par le noeud i. Nous avons donc besoin de calculer $\Delta\rho_{op}^{Noeud}(p)$ et donc de mesurer $\rho_{op}^{Noeud_i}(p)$ pour différentes valeurs de p afin d'être capable d'estimer $\Delta\rho_{op}^{Noeud}(p)$ pour un nombre p de processus exécutant l'opération op. Pour cela, nous devons être en mesure de savoir comment évolue $\Delta\rho_{op}^{Noeud}(p)$ en fonction de p (et ce pour chaque type de noeuds). Il s'agit d'une information que nous ne connaissons pas à priori.

Pour cela, nous nous appuyons sur quatre modèles possibles présentés dans le tableau ci-dessous :

linéaire	$\Delta\rho_{op}^{Noeud}(p) = \alpha p + \beta$
logarithmique	$\Delta\rho_{op}^{Noeud}(p) = \alpha ln(p) + \beta$
puissance	$\Delta\rho_{op}^{Noeud}(p) = \beta p^{\alpha}$
exponentiel	$\Delta\rho_{op}^{Noeud}(p) = \alpha^{p} + \beta$

Pour chaque type de noeuds, nous mesurons $\Delta\rho_{op}^{Noeud}(p)$ pour cinq valeurs de nombre de processus :

- le plus petit p possible noté p_{min}, c'est-à-dire 1 ;
- le plus grand p possible noté p_{max}, qui correspond au nombre de coeurs disponibles sur le noeud ;
- le nombre p médian noté p_2 qui correspond à la moitié du nombre de coeurs disponibles sur le noeud ;
- le nombre p_1 qui se situe au milieu de l'intervalle $[p_{min}\,;\,p_2]$;
- le nombre p_3 qui se situe au milieu de l'intervalle $[p_2\,;\,p_{max}]$

Ensuite, nous déterminons grâce à la méthode des moindres carrés [Rao et al., 1999] les coefficients (α et β) de chacun des quatre modèles en nous basant sur les cinq valeurs de $\Delta\rho_{op}^{Noeud}(p)$ mesurées. Nous calculons le coefficient de détermination R^2 correspondant à chacun des quatre modèles ajustés obtenus avec la méthode des moindres carrés. Nous considérons que $\Delta\rho_{op}^{Noeud}(p)$ évolue selon le modèle ajusté pour lequel le coefficient de détermination est le plus grand (*i.e.,* c'est-à-dire le plus proche de 1).

Pour nos mesures de $\Delta\rho_{op}^{Noeud}(p)$ (déduites grâce aux mesures de $\rho_{op}^{Noeud_i}(p)$), nous utilisons un wattmètre externe capable de nous fournir des mesures de puissances électriques moyennes avec une fréquence suffisamment élevée (1 Hz). Nous avons montré dans le chapitre précédent qu'OMEGAWATT respectait bien ces critères.

4.2.2 Calibration du temps d'exécution t_{op}

Selon l'opération considérée, t_{op} dépend d'un ou plusieurs paramètres. Nous supposons que l'on est capable d'identifier facilement ces paramètres grâce une connaissance des opérations (comme nous avons pu le faire pour les opérations identifiées liées à la tolérance aux panne et à la diffusion de données). Afin de prendre en compte les éventuels effets de congestion, nous considérons que le nombre p de processus du

même noeud effectuant simultanément la même opération *op* est un paramètre à prendre en compte pour notre calibration de t_{op}. Par exemple, ceci peut-être le cas si plusieurs processus d'un même noeud essaient d'écrire des données simultanément sur le disque dur local.

Pour calibrer t_{op}, nous avons besoin de mesurer le temps d'exécution en faisant varier les différents paramètres. Pour cela, nous mesurons t_{op} pour cinq valeurs uniformément réparties entre le minimum et le maximum possible pour chaque paramètre (tout en fixant tous les autres paramètres). Les cinq valeurs de chaque paramètre sont choisies de façon analogue à ce que nous avons présenté précédemment avec le paramètre p pour la calibration de $\Delta\rho_{op}^{Noeud}(p)$.

Deux cas se présentent :

1. Cas "avec modèle connu" : Soit nous connaissons un modèle selon lequel t_{op} évolue par rapport au(x) paramètre(s). Nous pouvons le connaître à partir de la littérature, grâce à la connaissance de l'algorithme utilisé dans l'opération *op* ou de la ressource sollicitée par l'opération *op*. Dans ce cas, nous déterminons les coefficients du modèle théorique grâce à la méthode des moindres carrés [Rao et al., 1999] en nous basant sur les cinq valeurs de chaque paramètre.
2. Cas "sans modèle connu" : Soit nous ne savons pas comment évolue t_{op} en fonction du (ou des) paramètre(s). Dans ce cas, pour chaque paramètre, nous procédons à la détermination du modèle ajusté par la méthode des moindres carrés en fonction de ce paramètre de façon analogue à ce que nous avons présenté pour la calibration de $\Delta\rho_{op}^{Noeud}(p)$, c'est-à-dire en nous appuyant sur les quatre modèles (linéaire, logarithmique, puissance et exponentiel).

Pour mesurer t_{op}, nous instrumentons le code de l'algorithme ou du protocole relatif à l'opération *op*, de manière à l'encadrer par deux instructions permettant de relever le temps de début et de fin de l'opération permettant ainsi de déduire le temps d'exécution correspondant. Afin de nous assurer que la calibration du temps d'exécution est précise, nous réalisons chaque mesure 30 fois et nous gardons seulement la valeur moyenne sur les 30 mesures.

4.2.2.1 Cas de la tolérance aux pannes

Dans cette section, nous décrivons les modèles utilisés pour les temps d'exécution de chaque opération des protocoles de tolérance aux pannes. Pour chaque opération *op*, t_{op} dépend de différents paramètres. Nous rappelons que la calibration de t_{op} n'est nécessaire que pour chaque type de noeuds. En d'autres termes, nous n'avons pas besoin de calibrer t_{op} sur tous les noeuds quand ils sont tous identiques.

Pour chaque type de noeuds, le temps $t_{sauvegarde}$ requis pour sauvegarder un point de reprise de taille V_{data} est :

$$t_{sauvegarde}(p, V_{data}) = t_{acces}(p) + t_{transfert}(p, V_{data}) = t_{acces}(p) + \frac{V_{data}}{r_{transfert}(p)}$$

De façon analogue, le temps $t_{enregistrement}$ requis pour enregistrer un message de

taille V_{data} est :

$$t_{enregistrement}(V_{data}) = t_{acces} + t_{transfert}(V_{data}) = t_{acces} + \frac{V_{data}}{r_{transfert}}$$

p est le nombre de processus au sein d'un même noeud qui essaient simultanément de sauvegarder des points de reprise. t_{acces} est le temps requis pour accéder au support de stockage où le point de reprise devra être sauvegardé ou bien le message enregistré. $t_{transfert}$ est le temps requis pour écrire une donnée de taille V_{data} sur le support de stockage. $r_{transfert}$ est le débit de transmission en écriture du support de stockage.

Dans le cas de la sauvegarde des points de reprise, t_{acces} et $r_{transfert}$ (et donc $t_{transfert}$) dépendent du nombre p de processus par noeud car les p processus sauvegardent leurs points de reprises simultanément sur le même support de stockage du fait que la fréquence de sauvegarde de points de reprise soit la même pour tous les processus d'une application. En ce qui concerne l'enregistrement de messages, un message n'est enregistré sur un support de stockage qu'une fois qu'il a été envoyé par un processus du noeud à travers l'interface réseau utilisée par le noeud. Ainsi, si plusieurs processus du noeud essaient d'envoyer des messages, il y aura une congestion au niveau de l'interface réseau et le temps d'envoi du message en cours recouvrera le temps d'enregistrement du message précédemment envoyé. En d'autres termes, cela signifie que nous considérons que l'on ne peut pas se retrouver dans un cas où plusieurs messages sont enregistrés simultanément par les p processus du noeud. Par conséquent, dans le cas de l'enregistrement de messages, t_{acces} et $r_{transfert}$ (et donc $t_{transfert}$) ne dépendent pas du nombre p de processus par noeud.

Comme expliqué dans la section 4.2.2, nous mesurons $t_{sauvegarde}$ en considérant les deux paramètres p et V_{data}. Nous disposons du modèle théorique de $t_{sauvegarde}$ en fonction de V_{data} donc pour ce paramètre, nous procédons à la détermination des coefficients du modèle théorique comme expliqué dans le cas "avec modèle connu" (section 4.2.2). Pour le paramètre p, nous ne disposons pas de modèle théorique donnant $t_{sauvegarde}$ en fonction de p et procédons donc comme cela est expliqué dans le cas "sans modèle connu" (section 4.2.2). En ce qui concerne $t_{enregistrement}$, il ne dépend que de V_{data} et nous disposons du modèle théorique donnant $t_{enregistrement}$ en fonction de ce paramètre. Nous procédons donc comme expliqué dans le cas "avec modèle connu".

Nous calibrons $t_{sauvegarde}$ et $t_{enregistrement}$ par rapport aux différents supports de stockage disponibles sur chaque noeud de la plate-forme (mémoire RAM, disque dur local, disque flash SSD, etc.).

Comme nous considérons uniquement la sauvegarde de points de reprise au niveau système, le protocole coordonné requiert une coordination entre tous les processus. Le temps d'une coordination entre tous les processus est :

$$t_{coordination}(N, p, V_{data}) = t_{attente}(V_{data}) + t_{synchro}(N, p) = \frac{V_{data}}{R_{transfert}} + t_{synchro}(N, p)$$

p est le nombre de processus du noeud i qui essaient d'effectuer la coordination. $t_{synchro}(N,p)$ est le temps requis pour échanger un marqueur de synchronisation entre tous les processus. $t_{synchro}(N,p)$ dépend du nombre de noeuds et du nombre de processus par noeud impliqués dans la synchronisation.

Nous ne disposons pas de modèle théorique pour $t_{synchro}(N,p)$ ni en fonction de N ni en fonction de p. Pour sa calibration, nous procédons donc comme cela est expliqué dans le cas "sans modèle connu" (section 4.2.2).

$t_{attente}(V_{data})$ est le temps nécessaire pour finir d'émettre les messages en cours de transmission au moment de la coordination. En d'autres termes, $t_{attente}(V_{data})$ est égal au temps requis pour transférer le message le plus volumineux de l'application. $R_{transfert}$ est le débit de transmission dans l'infrastructure réseau utilisée pour la plate-forme. En ce qui concerne $t_{attente}(V_{data})$, nous disposons d'un modèle théorique donnant $t_{attente}(V_{data})$. Pour sa calibration, nous procédons donc comme dans le cas "avec modèle connu" pour le paramètre V_{data}.

4.2.2.2 Cas de la diffusion de données

Dans cette section, nous décrivons les modèles de temps d'exécution que nous considérons pour chaque opération *op*. Pour chaque opération *op*, t_{op} dépend de plusieurs paramètres.

Dans [Wadsworth and Chen, 2008], les auteurs présentent des modèles théoriques pour les temps d'exécution des opérations *Scatter*, *AllGather* et *Pipepline*. Cependant ces modèles considèrent qu'il n'y a qu'un seul processus par noeud. Nous avons ajusté les modèles théoriques présentés dans [Wadsworth and Chen, 2008] afin de tenir compte du nombre p de processus par noeud (voir figure 4.1).

Ainsi, le temps requis pour effectuer un *Scatter*, un *AllGather* ou un *Pipepline* d'un volume de données V_{data} parmi N noeuds avec p processus par noeud est :

$$\begin{aligned} t_{Scatter}(N,p,V_{data}) &= t_{AllGather}(N,p,V_{data}) \\ &= (T_{Snet}(N,p) + \frac{V_{data}}{R_{net}(N,p)}) \cdot \frac{Np-1}{Np} \\ t_{Pipeline}(N,p,V_{data}) &= (T_{Snet}(N,p) + \frac{V_{data}}{R_{net}(N,p)}) \cdot \frac{C+Np-2}{C} \end{aligned}$$

$T_{Snet}(N,p)$ est le temps nécessaire pour démarrer le lien réseau et $R_{net}(N,p)$ est le débit de transfert pour transmettre un volume de données V_{data}. C est le nombre de pièces (de tailles égales) en lesquelles est découpée la donnée dans *MPI/Pipeline*. Il est égal au volume total de données à diffuser divisé par la taille de chaque pièce de donnée. Ainsi, la taille de chaque pièce et donc C dépend de l'implémentation choisie pour l'algorithme *MPI/Pipeline*.

En ce qui concerne $t_{Scatter}$, $t_{AllGather}$ et $t_{Pipeline}$, nous disposons des modèles théoriques en fonction des paramètres N, p et V_{data}. Par conséquent, pour la calibration des temps d'exécution de ces opérations, nous procédons comme expliqué dans le cas "avec modèle connu" (section 4.2.2).

Le temps nécessaire pour copier une donnée vers tous les processus se trouvant sur un même noeud est une fonction du nombre de processus par noeud p et du volume de données à copier V_{data} :

$$t_{CopyPrivate}(p, V_{data}) = t_{acces}(p) + \frac{V_{data}}{r_{transfert}(p)}$$

t_{acces} est le temps nécessaire pour accéder à la mémoire RAM dans laquelle la donnée V_{data} est présente suite à une opération de *AllGather* ou *Pipeline*. $r_{transfert}$ est le débit de transmission pour copier une donnée en mémoire RAM. Comme expliqué dans la section 4.2.2, nous mesurons $t_{CopyPrivate}$ en considérant les deux paramètres p et V_{data}. Nous disposons du modèle théorique de $t_{CopyPrivate}$ en fonction de V_{data} donc pour ce paramètre, nous procédons à la détermination des coefficients du modèle théorique comme expliqué dans le cas "avec modèle connu" (section 4.2.2). Nous ne disposons pas de modèle théorique donnant $t_{CopyPrivate}$ en fonction de p et procédons donc comme cela est expliqué dans le cas "sans modèle connu" (section 4.2.2).

4.3 Méthodologie d'estimation énergétique

Nous avons précédemment décrit comment nous réalisons la calibration énergétique. Une fois que le calibration est faite, l'estimateur est capable de fournir des estimations de l'énergie consommée par les différentes opérations identifiées dans chaque service étudié. La figure 4.2 montre les composants logiciels liés à l'estimation de la consommation énergétique.

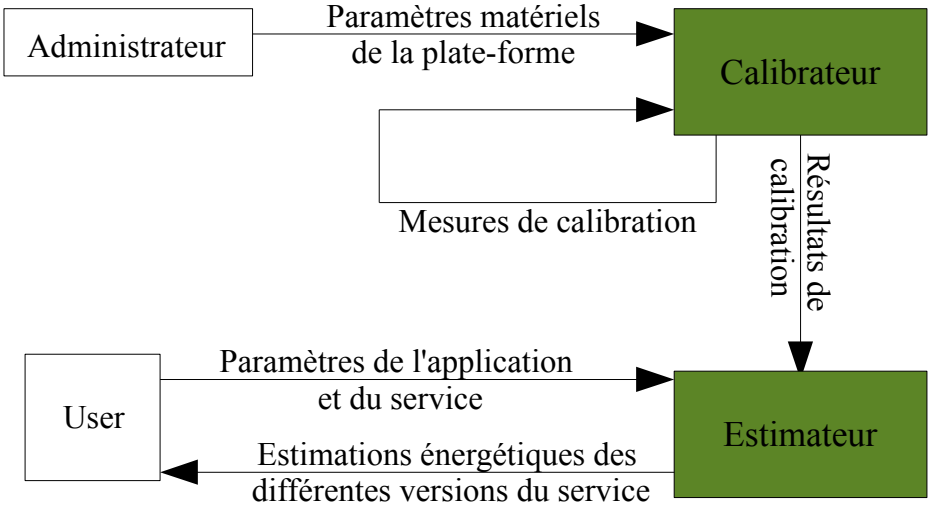

FIGURE 4.2 – Estimation de services applicatifs grâce à la calibration

Nous pouvons maintenant décrire comment estimer l'énergie consommée par chacune des opérations identifiées. Pour cela, nous détaillons les informations né-

cessaires : les paramètres fournis par l'utilisateur et les données mesurées par notre calibrateur.

Une fois que l'administrateur a fourni les paramètres matériels de la plate-forme, le calibrateur effectue les différentes mesures nécessaires pour constituer la base de connaissances sur les puissances électriques et les temps d'exécution des différentes opérations identifiées. Ensuite, en s'appuyant sur les résultats de calibration et sur une description de l'application (volume de données à diffuser, taille mémoire de l'application, etc.) et de l'environnement d'exécution (nombre de noeuds, liste des noeuds utilisés, nombre de processus par noeud, etc.) fournie par l'utilisateur, l'estimateur calcule la consommation énergétique des différentes versions du service applicatif.

Les paramètres que nous récupérons de l'utilisateur pour l'estimation dépendent de chacune des opérations à estimer. Dans le cas où ces paramètres correspondent à des valeurs que nous avons déjà mesurées lors de la calibration, l'estimation utilise directement ces valeurs pour calculer l'énergie consommée par l'opération en question. Si ce n'est pas le cas, c'est-à-dire s'il manque des points de mesure dans le calibrateur, l'estimateur s'appuie sur les modèles ajustés grâce à la méthode des moindres carrés [Rao et al., 1999] lors de la calibration. Dans les sections 4.3.1 et 4.3.2, nous détaillons respectivement, comment cette méthode s'applique au service de tolérance aux pannes et au service de diffusion de données.

4.3.1 Cas de la tolérance aux pannes

Cette section décrit comment nous estimons l'énergie consommée par chacune des opérations identifiées dans les protocoles de tolérance aux panne. Pour cela, nous détaillons les informations nécessaires : les paramètres fournis par l'utilisateur et les données mesurées par notre calibrateur.

4.3.1.1 Sauvegarde de points de reprise

Afin d'estimer la consommation énergétique de la sauvegarde de points de reprise, l'estimateur récupère de l'utilisateur la taille mémoire totale requise par l'application à exécuter, le nombre N de noeuds et le nombre p de processus par noeud. À partir de ces informations, l'estimateur calcule la taille mémoire moyenne $V_{memoire}^{moyen}$ requise par chaque processus (taille mémoire totale divisée par le nombre de processus). Ensuite, l'estimateur récupère du calibrateur le surcoût moyen de puissance électrique $\Delta\rho_{sauvegarde}(p)$, le temps $t_{sauvegarde}(p, V_{memoire}^{moyen})$ selon les modèles ajustés grâce à la méthode des moindres carrés dans l'étape de la calibration. Il récupère également la mesure $\rho_{repos}^{Noeud_i}$ pour chaque noeud i. Nous notons respectivement par $\xi_{sauvegarde}^{Noeud_i}(p)$ et $\rho_{sauvegarde}^{Noeud_i}(p)$ la consommation énergétique et la puissance électrique moyenne de chaque noeud i réalisant une sauvegarde de points de reprise. L'estimation de la consommation énergétique d'une seule sauvegarde de points de reprise est donnée

par :

$$
\begin{aligned}
E_{sauvegarde} &= \sum_{i=1}^{N} \xi_{sauvegarde}^{Noeud_i}(p) \\
&= \sum_{i=1}^{N} \rho_{sauvegarde}^{Noeud_i}(p) \cdot t_{sauvegarde}(p, V_{memoire}^{moyen})) \\
&= t_{sauvegarde}(p, V_{memoire}^{moyen}) \cdot \sum_{i=1}^{N} (\rho_{repos}^{Noeud_i} + \Delta\rho_{sauvegarde}(p)) \\
&= t_{sauvegarde}(p, V_{memoire}^{moyen}) \cdot (N \cdot \Delta\rho_{sauvegarde}(p) + (\sum_{i=1}^{N} \rho_{repos}^{Noeud_i}))
\end{aligned}
$$

4.3.1.2 Enregistrement de messages

Pour estimer la consommation énergétique de l'enregistrement de messages, l'estimateur récupère de l'utilisateur le nombre N de noeuds, le nombre p de processus par noeud, le nombre et la taille totale de l'ensemble des messages envoyés pendant l'application qu'il souhaite exécuter. Grâce à ces informations, l'estimateur calcule le volume moyen de données V_{data}^{moyen} envoyées et donc enregistrées sur chaque noeud (taille totale de l'ensemble des messages envoyés divisée par le nombre N de noeuds). Ensuite, l'estimateur récupère du calibrateur le surcoût moyen de puissance électrique $\Delta\rho_{enregistrement}$, le temps $t_{enregistrement}(p, V_{data}^{moyen})$ selon les modèles ajustés grâce à la méthode des moindres carrés dans l'étape de la calibration. Il récupère également la mesure $\rho_{repos}^{Noeud_i}$ pour chaque noeud i.

L'estimation de la consommation énergétique de l'opération d'enregistrement de messages est donnée par :

$$
\begin{aligned}
E_{enregistrement} &= \sum_{i=1}^{N} \xi_{enregistrement}^{Noeud_i}(p) \\
&= \sum_{i=1}^{N} \rho_{enregistrement}^{Noeud_i}(p) \cdot t_{enregistrement}(V_{data}^{moyen}) \\
&= t_{enregistrement}(V_{data}^{moyen}) \cdot \sum_{i=1}^{N} (\rho_{repos}^{Noeud_i} + \Delta\rho_{enregistrement}(p)) \\
&= t_{enregistrement}(V_{data}^{moyen}) \cdot (N \cdot \Delta\rho_{enregistrement}(p) + (\sum_{i=1}^{N} \rho_{repos}^{Noeud_i}))
\end{aligned}
$$

4.3.1.3 Coordination

Nous rappelons que la coordination se décompose en deux phases : la phase d'attente d'envoi des messages en cours suivie de la phase de synchronisation de tous les processus. Pour estimer la consommation énergétique de la coordination,

l'estimateur calcule la taille moyenne d'un message $V_{message}^{moyen}$ comme la taille totale des messages divisée par le nombre total de messages échangés. L'estimateur utilise aussi le nombre total de noeuds N et le nombre de processus par noeuds p. Ensuite, l'estimateur récupère du calibrateur le surcoût moyen de puissance électrique $\Delta\rho_{synchro}(p)$, le temps $t_{synchro}(N,p)$ selon les modèles ajustés grâce à la méthode des moindres carrés dans l'étape de la calibration. Il récupère également la mesure $\rho_{repos}^{Noeud_i}$ pour chaque noeud i. L'estimation de la consommation énergétique $E_{synchro}$ d'une synchronisation est donnée par :

$$\begin{aligned} E_{synchro} &= \sum_{i=1}^{N} \xi_{synchro}^{Noeud_i}(N,p) \\ &= \sum_{i=1}^{N} \rho_{synchro}^{Noeud_i}(p) \cdot t_{synchro}(N,p) \\ &= t_{synchro}(N,p) \cdot \sum_{i=1}^{N} (\rho_{repos}^{Noeud_i} + \Delta\rho_{synchro}(p)) \\ &= t_{synchro}(N,p) \cdot (N \cdot \Delta\rho_{synchro}(p) + (\sum_{i=1}^{N} \rho_{repos}^{Noeud_i})) \end{aligned}$$

En ce qui concerne l'attente active, l'estimateur récupère du calibrateur le surcoût moyen de puissance électrique $\Delta\rho_{attente}(p)$, le temps $t_{attente}(N,p,V_{message}^{moyen})$ selon les modèles ajustés grâce à la méthode des moindres carrés dans l'étape de la calibration. L'estimation de la consommation énergétique $E_{attente}$ d'une attente active est donnée par :

$$\begin{aligned} E_{attente} &= \sum_{i=1}^{N} \xi_{attente}^{Noeud_i}(N,p) \\ &= \sum_{i=1}^{N} \rho_{attente}^{Noeud_i}(p) \cdot t_{attente}(V_{message}^{moyen}) \\ &= t_{attente}(V_{message}^{moyen}) \cdot \sum_{i=1}^{N} (\rho_{repos}^{Noeud_i} + \Delta\rho_{attente}(p)) \\ &= t_{attente}(V_{message}^{moyen}) \cdot (N \cdot \Delta\rho_{attente}(p) + (\sum_{i=1}^{N} \rho_{repos}^{Noeud_i})) \end{aligned}$$

L'estimateur évalue alors la consommation énergétique d'une coordination comme suit :

$$E_{coordination} = E_{attente} + E_{synchro}$$

4.3.2 Cas de la diffusion de données

Cette section décrit comment nous estimons l'énergie consommée par chacune des opérations identifiées dans les algorithmes de diffusion de données. Pour cela,

nous détaillons notamment les informations nécessaires : les paramètres fournis par l'utilisateur et les données mesurées par notre calibrateur.

4.3.2.1 MPI/SAG et Hybrid/SAG

Pour estimer la consommation énergétique de *MPI/SAG* et de *Hybrid/SAG*, l'estimateur récupère de l'utilisateur le nombre de noeuds N, le nombre p de processus par noeud, et la taille des données à diffuser V_{data}. À partir de ces informations, l'estimateur récupère du calibrateur $\Delta\rho_{Scatter}(p)$, $\Delta\rho_{AllGather}(p)$, $\Delta\rho_{CopyPrivate}(p)$, $t_{Scatter}(N,p,V_{data})$, $t_{AllGather}(N,p,V_{data})$ et $t_{CopyPrivate}(p,V_{data})$ selon les modèles ajustés grâce à la méthode des moindres carrés dans l'étape de la calibration. Par ailleurs, l'estimateur récupère auprès du calibrateur le nombre de commutateurs réseau M en se basant sur la description matérielle de la plate-forme faite par l'administrateur.

L'estimation de la consommation énergétique de l'algorithme *MPI/SAG* est donnée par :

$$\begin{aligned}
E_{MPI/SAG} &= \sum_{i=1}^{N} \xi_{MPI/SAG}^{Noeud_i} + \sum_{j=1}^{M} \xi_{MPI/SAG}^{Switch_j} \\
&= t_{Scatter}(N,p,V_{data}) \cdot (\sum_{i=1}^{N} \rho_{Scatter}^{Noeud_i}(p) + \sum_{j=1}^{M} \rho_{Scatter}^{Switch_j}) \\
&+ t_{AllGather}(N,p,V_{data}) \cdot (\sum_{i=1}^{N} \rho_{AllGather}^{Noeud_i}(p) + \sum_{j=1}^{M} \rho_{AllGather}^{Switch_j})
\end{aligned}$$

L'estimation de la consommation énergétique de l'algorithme *Hybrid/SAG* est donnée par :

$$\begin{aligned}
E_{Hybrid/SAG} &= \sum_{i=1}^{N} \xi_{Hybrid/SAG}^{Noeud_i} + \sum_{j=1}^{M} \xi_{Hybrid/SAG}^{Switch_j} \\
&= t_{Scatter}(N,1,V_{data}) \cdot (\sum_{i=1}^{N} \rho_{Scatter}^{Noeud_i}(1) + \sum_{j=1}^{M} \rho_{Scatter}^{Switch_j}) \\
&+ t_{AllGather}(N,1,V_{data}) \cdot (\sum_{i=1}^{N} \rho_{AllGather}^{Noeud_i}(1) + \sum_{j=1}^{M} \rho_{AllGather}^{Switch_j}) \\
&+ t_{CopyPrivate}(p,V_{data}) \cdot \sum_{i=1}^{N} (\rho_{CopyPrivate}^{Noeud_i}(p))
\end{aligned}$$

4.3.2.2 MPI/Pipeline et Hybrid/Pipeline

Par rapport à l'estimation de *MPI/SAG* et de *Hybrid/SAG*, l'estimateur récupère de l'utilisateur un paramètre complémentaire qui est la taille de chaque pièce (*chunk*) et ce dans l'optique d'estimer la consommation énergétique de *MPI/Pipeline* et de *Hybrid/Pipeline*. Ce paramètre est nécessaire pour la détermination

du temps d'exécution de *Pipeline* dans la mesure où $t_{Pipeline}(N, p, V_{data})$ dépend de la constante C. À partir du calibrateur, l'estimateur récupère $\Delta\rho_{Pipeline}(p)$ et $t_{Pipeline}(N, p, V_{data})$. L'estimation de la consommation énergétique de l'algorithme *MPI/Pipeline* est donnée par :

$$\begin{aligned} E_{MPI/Pipeline} &= \sum_{i=1}^{N} \xi_{MPI/Pipeline}^{Noeud_i} + \sum_{j=1}^{M} \xi_{MPI/Pipeline}^{Switch_j} \\ &= t_{Pipeline}(N, p, V_{data}) \cdot (\sum_{i=1}^{N} \rho_{Pipeline}^{Noeud_i}(p) + \sum_{j=1}^{M} \rho_{Pipeline}^{Switch_j}) \end{aligned}$$

L'estimation de la consommation énergétique de l'algorithme *Hybrid/Pipeline* est donnée par :

$$\begin{aligned} E_{Hybrid/Pipeline} &= \sum_{i=1}^{N} \xi_{Hybrid/Pipeline}^{Noeud_i} + \sum_{j=1}^{M} \xi_{Hybrid/Pipeline}^{Switch_j} \\ &= t_{Pipeline}(N, 1, V_{data}) \cdot (\sum_{i=1}^{N} \rho_{Pipeline}^{Noeud_i}(1) + \sum_{j=1}^{M} \rho_{Pipeline}^{Switch_j}) \\ &+ \sum_{i=1}^{N} (t_{CopyPrivate}(p, V_{data}) \cdot \rho_{CopyPrivate}^{Noeud_i}(p)) \end{aligned}$$

4.4 Validation des estimations

Pour valider nos estimations, nous exécutons différentes applications réelles de calcul haute performance avec les différents protocoles de tolérance aux pannes ou différents scénarios de diffusion de données sur une grappe homogène de la plate-forme expérimentale distribuée à large échelle Grid'5000 [Cappello et al., 2005] puis nous comparons la consommation énergétique mesurée réellement à la consommation énergétique évaluée grâce à notre estimateur. Pour les expérimentations visant à valider nos estimations, nous avons utilisé la grappe *Taurus* de Grid'5000. Cette grappe est décrite dans le tableau 3.1 qui se trouve dans la section 3.1.1 du chapitre précédent.

Nous utilisons les mêmes notations que les sections précédentes : N est le nombre de noeuds, p est le nombre de processus et op désigne une des opérations identifiées (sauvegarde de points de reprise, *Pipeline*, etc.).

4.4.1 Résultats de calibration de la plate-forme

Dans cette section, nous présentons une partie des résultats de calibration de la plate-forme selon la méthodologie décrite dans la section 4.2. La grappe *Taurus* n'est

composée que de noeuds identiques : donc il n'y a qu'un seul type de noeuds. Ils sont tous interconnectés à l'aide d'un seul commutateur réseau.

4.4.1.1 Calibration de la puissance électrique

Tout d'abord, nous mesurons la puissance électrique au repos ρ^i_{repos} de chaque noeud i de la grappe *Taurus*. La figure 4.3 montre la puissance électrique au repos des 16 noeuds appartenant à la grappe *Taurus*. Á partir de cette figure et même si la grappe est composée de noeuds homogènes, nous notons le besoin de calibrer la puissance électrique au repos de chaque noeud.

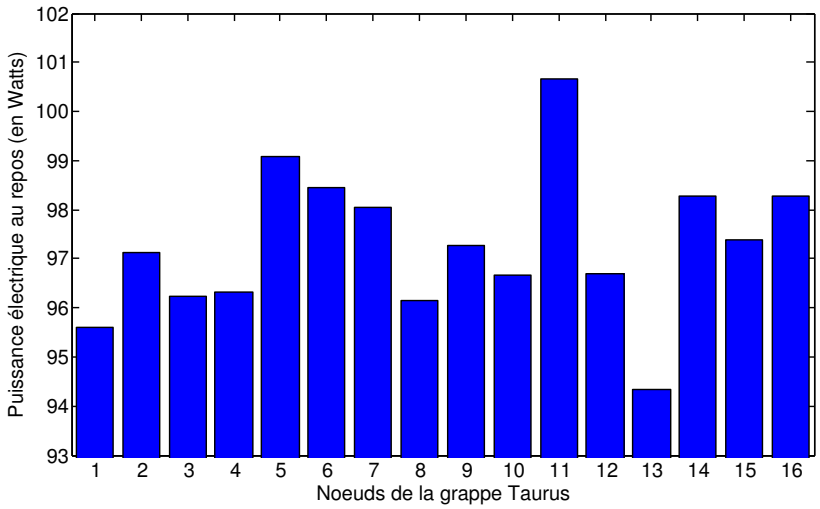

FIGURE 4.3 – Puissance électrique au repos (moyennée) des noeuds de la grappe *Taurus*

Pour chacune des opérations *op* identifiées et pour chaque type de noeuds de la grappe, nous calibrons avec OMEGAWATT le surcoût moyen de puissance électrique liée à l'opération *op*, $\Delta\rho_{op}(p)$, comme expliqué dans la section 4.2.1. Étant donné que chaque noeud de la grappe *Taurus* comporte 12 coeurs de calcul, les cinq valeurs de p que nous choisissons pour la calibration de $\Delta\rho_{op}(p)$ sont 1, 4, 6, 9 et 12 processus par noeud.

Les figures 4.4 et 4.5 montrent les mesures de $\Delta\rho_{op}(p)$ pour les cinq valeurs de p et pour chaque opération *op* identifiée respectivement dans les protocoles de tolérance aux pannes et dans les algorithmes de diffusion de données.

Nous notons sur les figures 4.4 et 4.5 que pour certaines opérations, $\Delta\rho_{op}(p)$ varie bien en fonction du nombre de coeurs par noeud qui effectuent la même opération. Pour certaines opérations comme la sauvegarde de points de reprises, $\Delta\rho_{op}(p)$ reste quasiment constant en fonction de p. Pour le $\Delta\rho_{op}(p)$ de ces opérations, nous obtenons un des quatre modèles du calibrateur (voir section 4.2.1) avec un coefficient α très proche de 0 et une valeur de β très proche de la valeur constante de

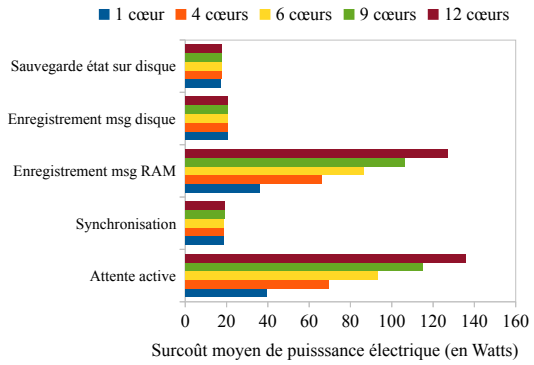

FIGURE 4.4 – Surcoût moyen de puissance électrique des opérations liées aux protocoles de tolérance aux pannes

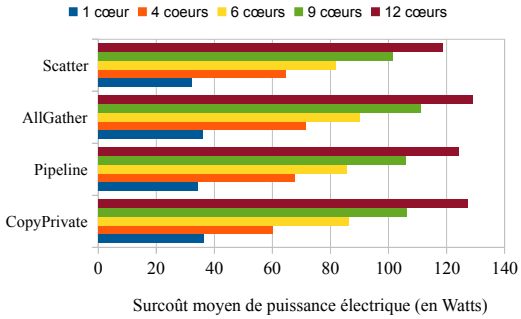

FIGURE 4.5 – Surcoût moyen de puissance électrique des opérations liées aux algorithmes de diffusion de données

$\Delta\rho_{op}(p)$ (*i.e.,* c'est-à-dire des modèles quasi-stationnaires). Par exemple, le modèle de $\Delta\rho_{sauvegarde}(p)$ ajusté par la méthode des moindres carrés à partir de ces cinq valeurs de p est :

$$\Delta\rho_{sauvegarde}(p) = 17.22 \cdot p^{0.0084271}$$

Bien que le modèle ajusté soit un modèle puissance, le coefficient α très faible implique que $\Delta\rho_{sauvegarde}(p)$ est quasi-stationnaire en fonction de p. Le coefficient de détermination R^2 correspondant à ce modèle est de 0.976, ce qui est très proche de 1.

Pour d'autres opérations telle que *Pipeline*, $\Delta\rho_{op}(p)$ augmente en fonction de p. À titre d'exemple, le modèle de $\Delta\rho_{Pipeline}(p)$ obtenu lors de la calibration est :

$$\Delta\rho_{Pipeline}(p) = 33.85 \cdot p^{0.518956}$$

Le fait que α soit très proche de 0.5 signifie que $\Delta\rho_{Pipeline}(p)$ est quasiment exprimé en fonction de $\sqrt{p}$. Le coefficient de détermination de R^2 correspondant à ce modèle est de 0.999, ce est également très proche de 1.

Par ailleurs, nous mesurons la consommation au repos d'un commutateur réseau 10 Gigabit Ethernet pendant 300 secondes suivie de sa puissance électrique pendant un trafic réseau intense durant 300 secondes. Afin de mesurer sa puissance électrique au repos, nous nous assurons qu'il n'y a aucun trafic réseau, et ce en éteignant tous les noeuds qui sont interconnectés par le commutateur réseau. Pour mesurer sa puissance électrique pendant un trafic réseau intense, nous lançons `iperf` en mode serveur sur un des noeuds et `iperf` en mode client sur tous les autres noeuds interconnectés par le commutateur réseau. La figure 4.6 montre les mesures électriques obtenues.

D'après la figure 4.6, nous remarquons que la puissance électrique du commutateur reste quasiment constante pendant toute la durée de l'expérience. En d'autres termes, la puissance électrique du commutateur réseau ne varie pas en fonction du

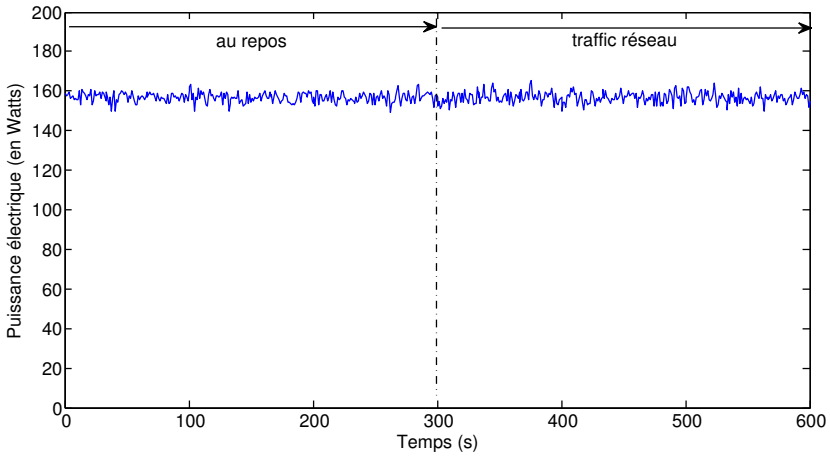

FIGURE 4.6 – Puissance électrique du commutateur au repos pendant 300 secondes puis avec une charge réseau intense durant 300 secondes

trafic réseau. Cela signifie que $\Delta\rho_{op}^{Switch}$ est (quasiment) égal à 0 pour tous les opérations ($\forall op, \rho_{op}^{Switch_j} = \rho_{idle}^{Switch_j}$). Une étude récente [Mahadevan et al., 2009, Hlavacs et al., 2009] confirme ce fait en évaluant et en montrant que la puissance électrique de plusieurs équipements réseau n'est pas influencée par le trafic réseau. Ceci dit, même dans le cas où la puissance électrique d'un commutateur réseau dépendrait du trafic réseau, notre approche de calibration permettrait de le prendre en compte en mesurant $\Delta\rho_{op}^{Switch}$ pour chaque opération op.

4.4.1.2 Calibration du temps d'exécution

En nous appuyant sur la méthodologie présentée dans la section 4.2.2, nous calibrons le temps d'exécution pour chaque opération sur chaque type de noeuds la plate-forme expérimentale.

4.4.1.2.1 Cas de la tolérance aux pannes

Afin de calibrer le temps d'exécution de la sauvegarde de points de reprise sur disque dur local, nous considérons un nombre variable de coeurs par noeud qui sauvegardent simultanément des points de reprise et nous mesurons ce temps pour différentes tailles de points de reprise V_{data} pour un noeud de la plate-forme expérimentale. Chaque processus du noeud sauvegarde un point de reprise de taille V_{data}. En d'autres termes, lorsqu'il y a p processus qui sauvegardent simultanément des points de reprise, un volume de $p \cdot V_{data}$ est sauvegardé sur le disque dur local. La figure 4.7 présente les temps pour sauvegarder des points de reprise mesurés sur un noeud de la plate-forme expérimentale. Comme expliqué dans 4.2.2, nous choisissons 1, 4, 6, 9 et 12 processus par noeud pour les cinq valeurs de p et 0 Mo, 500 Mo, 1000 Mo, 1500 Mo et 2000 Mo pour les cinq valeurs de V_{data}. Le choix de 2000 Mo comme taille maximale de point de reprise est motivé par le fait que chaque noeud n'a que 32 Go de mémoire susceptibles d'être

partagés par les 12 coeurs de calcul. Pour les différentes valeurs de p, la figure montre comment évolue $t_{sauvegarde}$ en fonction de V_{data}.

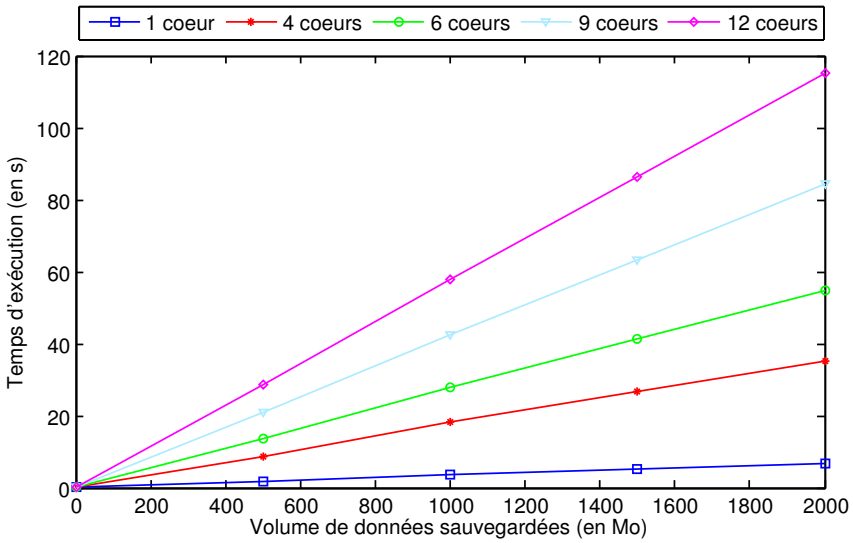

FIGURE 4.7 – Calibration de la sauvegarde de points de reprise sur disque dur local

D'abord, nous observons que les courbes ont une tendance linéaire en fonction de V_{data} pour p fixé. Par exemple pour $p = 4$, le modèle de $t_{sauvegarde}$ ajusté par la méthode des moindres carrés [Rao et al., 1999] à partir des cinq valeurs de V_{data} :

$$t_{sauvegarde}(4, V_{data}) = \frac{1}{0.56569 \cdot 10^9} \cdot V_{data} + 0.09433 \cdot 10^{-3}$$

Nous notons également que pour V_{data} fixé, $t_{sauvegarde}$ augmente lorsque p croît et ce à cause de la congestion d'entrées-sorties générée par les accès concurrents par les p processus sur le disque dur local. Pour $V_{data} = 1000Mo$, le modèle de $t_{sauvegarde}$ ajusté par la méthode des moindres carrés à partir des cinq valeurs de p :

$$t_{sauvegarde}(p, 1000Mo) = 4.91359 \cdot p - 1.5026$$

Si par exemple, nous voulons estimer le temps $t_{sauvegarde}(3, 800Mo)$, c'est-à-dire pour des valeurs de p et de V_{data} qui ne sont pas tous les deux parmi les cinq valeurs mesurées, nous calculons :

- d'une part : $t_{sauvegarde}(1, 800Mo)$, $t_{sauvegarde}(4, 800Mo)$, $t_{sauvegarde}(6, 800Mo)$, $t_{sauvegarde}(9, 800Mo)$ et $t_{sauvegarde}(12, 800Mo)$, respectivement à partir des équations $t_{sauvegarde}(1, V_{data})$, $t_{sauvegarde}(4, V_{data})$, $t_{sauvegarde}(6, V_{data})$, $t_{sauvegarde}(9, V_{data})$ et $t_{sauvegarde}(12, V_{data})$;
- d'autre part : $t_{sauvegarde}(3, 0Mo)$, $t_{sauvegarde}(3, 500Mo)$, $t_{sauvegarde}(3, 1000Mo)$, $t_{sauvegarde}(3, 1500Mo)$, $t_{sauvegarde}(3, 2000Mo)$, respectivement à partir des équations $t_{sauvegarde}(p, 0Mo)$, $t_{sauvegarde}(p, 500Mo)$, $t_{sauvegarde}(p, 1000Mo)$, $t_{sauvegarde}(p, 1500Mo)$, $t_{sauvegarde}(p, 2000Mo)$.

À partir des valeurs calculées $t_{sauvegarde}(1, 800Mo)$, $t_{sauvegarde}(4, 800Mo)$, $t_{sauvegarde}(6, 800Mo)$, $t_{sauvegarde}(9, 800Mo)$ et $t_{sauvegarde}(12, 800Mo)$, nous déterminons par la méthode des moindres carrées, le modèle donnant $t_{sauvegarde}(p, 800Mo)$ en fonction de p (comme expliqué dans la section 4.2.2) et calculons le coefficient de détermination R^2 correspondant au modèle ainsi ajusté. De même, à partir des valeurs $t_{sauvegarde}(3, 0Mo)$, $t_{sauvegarde}(3, 500Mo)$, $t_{sauvegarde}(3, 1000Mo)$, $t_{sauvegarde}(3, 1500Mo)$, $t_{sauvegarde}(3, 2000Mo)$, nous déterminons le modèle donnant $t_{sauvegarde}(3, V_{data})$ en fonction de V_{data} et calculons le coefficient de détermination R^2 correspondant au modèle ainsi ajusté. Ensuite, entre $t_{sauvegarde}(p, 800Mo)$ et $t_{sauvegarde}(3, V_{data})$, nous choisissons le modèle pour lequel le coefficient de détermination est le plus proche de 1. Enfin nous calculons $t_{sauvegarde}(3, 800Mo)$ en fonction du modèle obtenu.

Nous présentons sur la figure 4.8, les mesures obtenues du temps d'exécution pour l'enregistrement de messages que ce soit dans la mémoire RAM ou sur le disque dur. Afin de calibrer le temps d'exécution de l'enregistrement de messages sur mémoire ou sur disque, nous mesurons ce temps pour différentes tailles de message V_{data} pour un noeud de la plate-forme expérimentale. Les valeurs choisies pour V_{data} sont 0 Ko, 500 Ko, 1000 Ko, 1500 Ko et 2000 Ko. Comme expliqué dans la section 4.2.2.1, nous n'avons pas besoin de calibrer $t_{enregistrement}$ en fonction de p car les processus n'enregistrent pas simultanément leurs messages envoyés sur le support de stockage du fait que l'envoi de message soit congestionné par le partage d'une seule interface réseau par noeud. Nous mesurons donc ces temps lorsqu'un seul processus du noeud ($p = 1$) exécute l'opération d'enregistrement de messages.

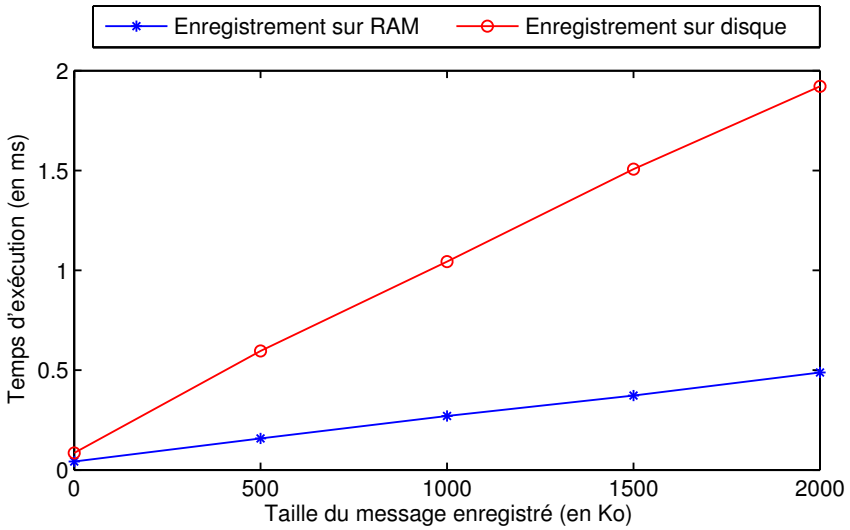

FIGURE 4.8 – Calibration de l'enregistrement de messages sur mémoire RAM et sur disque dur local

Nous observons que les courbes ont une tendance linéaire et ce aussi bien pour l'enregistrement de messages sur mémoire RAM que sur disque dur local. Le temps

d'enregistrement de messages sur disque dur local est plus élevé que celui de la mémoire RAM et ce quelque soit la taille du message enregistré.

De façon analogue à la sauvegarde de points de reprise, nous obtenons les modèles ajustés suivants pour $t_{enregistrement}$:

$$\begin{array}{lcl} \text{Dans la mémoire RAM} & : & t_{enregistrement}(V_{data}) = \frac{1}{4.4342 \cdot 10^9} \cdot V_{data} + 0.0426 \cdot 10^{-3} \\ \text{Sur le disque dur local} & : & t_{enregistrement}(V_{data}) = \frac{1}{1.0552 \cdot 10^9} \cdot V_{data} + 0.0858 \cdot 10^{-3} \end{array}$$

En ce qui concerne la coordination, nous avons besoin de calibrer le temps de la synchronisation ainsi que le temps de transfert d'un message.

Afin de calibrer le temps de synchronisation $t_{synchro}(N, p)$ des Np processus, nous mesurons ce temps pour différentes valeurs de N et pour différentes valeurs de p. Les valeurs mesurées pour p et N sont choisies comme nous l'avons expliqué dans la section 4.2.2. Dans notre cas, les valeurs mesurées pour p sont 1, 4, 6, 9 et 12 alors que les valeurs mesurées pour N sont 1, 4, 8, 12 et 16. La figure 4.9 présente les temps de synchronisation mesurés par le calibrateur. Par exemple, le point 4 coeurs/8 noeuds est le temps requis pour synchroniser 32 processus uniformément répartis sur 8 noeuds. D'abord, nous constatons que le temps pour synchroniser des processus se trouvant sur un même noeud est plus faible que pour des processus se trouvant sur des noeuds distincts. En effet, cela nécessite beaucoup moins de temps pour synchroniser des processus se trouvant sur un même noeud que pour des processus se trouvant sur des noeuds distincts. Le débit de transmission sur le réseau est beaucoup plus bas que le débit de transmission au sein d'un même noeud.

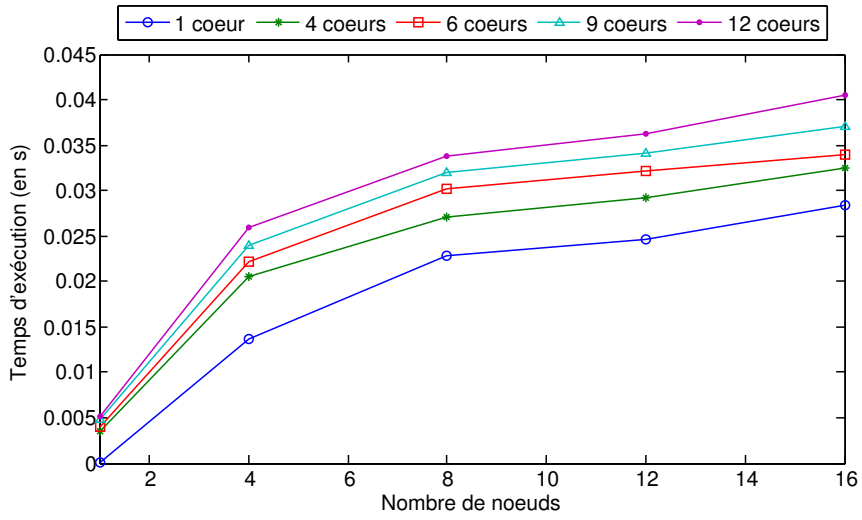

FIGURE 4.9 – Calibration du temps de synchronisation de la plate-forme expérimentale

Par exemple pour $p = 4$, le modèle de $t_{synchro}$ ajusté par la méthode des moindres

carrés à partir des cinq valeurs de N est :

$$t_{synchro}(N, 4) = 0.0103757 \cdot ln(N) + 0.00445945$$

Pour $N = 8$, le modèle de $t_{synchro}$ ajusté par la méthode des moindres carrés à partir des cinq valeurs de p est :

$$t_{synchro}(8, p) = 0.00443799 \cdot ln(p) + 0.02225942$$

Si par exemple, nous voulons estimer le temps $t_{synchro}(N, p)$, c'est-à-dire pour des valeurs de N et de p qui ne sont pas tous les deux parmi les cinq valeurs mesurées, alors nous procédons de façon analogue à ce que nous l'avons expliqué pour $t_{sauvegarde}(p, V_{data})$.

Nous calibrons le temps nécessaire pour transférer un message (*i.e.,* de l'attente active ayant lieu au moment de la coordination) sur la plate-forme expérimentale en faisant varier la taille V_{data} du message à transférer. Pour cela, nous mesurons le temps $t_{attente}(V_{data})$ de transfert d'un message émis à l'aide de *MPI_Send* par un processus se trouvant sur un noeud donné à destination d'un processus se trouvant sur un autre noeud. Dans le cas général, il faut faire cette mesure pour chaque couple de processus se trouvant à différents niveaux de la hiérarchie réseau, c'est à dire pour deux processus qui n'ont besoin de traverser qu'un seul commutateur réseau, puis pour deux processus qui ont besoin de traverser deux commutateurs réseaux, etc. Dans notre plate-forme expérimentale, un seul commutateur réseau interconnecte tous les noeuds donc nous n'avons besoin de mesurer ce temps que pour un couple de processus sur des noeuds différents.

Afin de calibrer le temps d'exécution du transfert de message sur le réseau, nous mesurons ce temps pour différentes tailles de message V_{data} pour un couple de processus se trouvant sur deux noeuds distincts. Les valeurs choisies pour V_{data} sont 0 Ko, 500 Ko, 1000 Ko, 1500 Ko et 2000 Ko. Sur la figure 4.10, nous présentons la calibration du temps de transfert d'un message.

Les temps de transfert mesurés dépendent linéairement de la taille du message à transférer. De façon analogue à la sauvegarde de points de reprise, nous obtenons le modèle ajusté suivant pour $t_{attente}$:

$$t_{attente}(V_{data}) = \frac{1}{0.60148 \cdot 10^9} \cdot V_{data} + 3.6222 \cdot \ 10^{-3}$$

4.4.1.2.2 Cas de la diffusion de données

Afin de calibrer le temps d'exécution des différentes opérations ($t_{Scatter}(N, p, V_{data})$, $t_{AllGather}(N, p, V_{data})$ et $t_{Pipeline}(N, p, V_{data})$), nous mesurons ces temps d'exécutions pour différentes valeurs de N, de p et de V_{data}. Comme expliqué dans la section 4.2.2.2, étant donné que pour ces trois opérations nous avons un modèle théorique exprimé en fonction de tous les paramètres (N, p et V_{data}), nous déterminons par la méthode des moindres carrées, les coefficients $T_{Snet}(N, p)$ et $R_{net}(N, p)$ que nous retrouvons dans chacun des trois modèles.

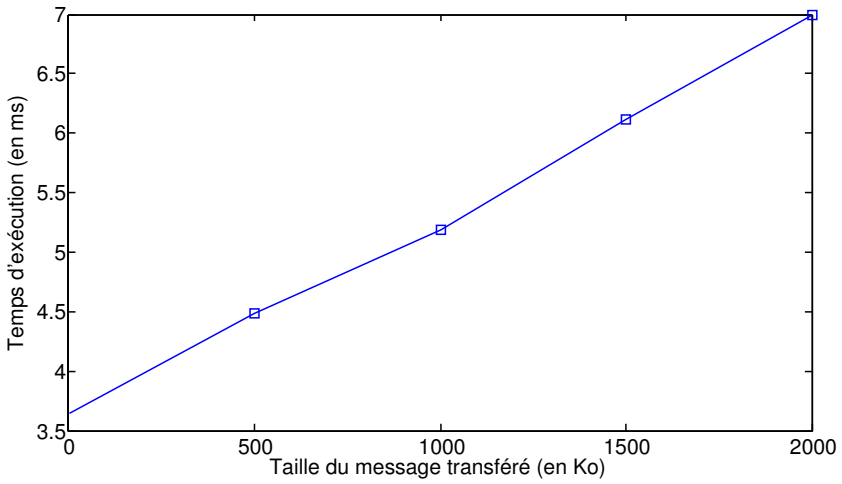

FIGURE 4.10 – Calibration de l'attente active pour la plate-forme expérimentale

Nous présentons une partie des résultats de calibration des temps d'exécution pour les opérations *Scatter* et *Pipeline* sur la figure 4.11, pour un nombre variable de noeuds (4, 8, 12, 16), pour un processus par noeud et pour un volume de données allant de 0 Mo à 2000 Mo en passant par les valeurs 500 Mo, 1000 Mo et 1500 Mo (comme expliqué dans la section 4.2.2). Sur cette figure, nous n'avons pas affiché les mesures de temps d'exécution pour l'opération *AllGather* car les courbes obtenues pour cette opération sont superposables à ceux de l'opération *Scatter* du fait que les modèles théoriques soient les mêmes pour ces deux opérations (voir section 4.2.2.2).

Afin de pas surcharger la figure 4.11 (25x2 courbes au lieu de 5x2 car 2 opérations représentées), nous n'avons présenté les mesures que pour un nombre fixé de processus par noeud. Le choix de présenter ces mesures pour un nombre p égal à 1 est motivé par le fait que nous utilisons également $t_{Scatter}(N, 1, V_{data})$, $t_{AllGather}(N, 1, V_{data})$ et $t_{Pipeline}(N, 1, V_{data})$ pour l'estimation énergétique des diffusions hybrides (ce qui n'est pas le cas pour $p > 1$).

Pour N et p fixés, la figure 4.11 montre que $t_{Scatter}$ et $t_{Pipeline}$ évoluent linéairement par rapport à V_{data}. En d'autres termes, $T_{Snet}(N, p)$ et $R_{net}(N, p)$ sont constants si on considère un nombre fixé de noeuds et un nombre donné de processus par noeud. Pour un seul paramètre fixé p (ou N), nous déterminons $T_{Snet}(N, p)$ et $R_{net}(N, p)$ par la méthode des moindres carrés $T_{Snet}(N, p)$ et $R_{net}(N, p)$ en fonction de l'autre paramètre N (ou p), de façon analogue à ce que l'on a présenté pour $t_{sauvegarde}(p, V_{data})$. C'est grâce à ces modèles que nous estimons T_{Snet} et R_{net} pour des valeurs de N et p données par l'utilisateur lorsqu'un des deux paramètres fait partie des cinq valeurs mesurées. Lorsqu'aucun des deux paramètres n'a été mesuré par notre calibrateur, nous procédons de façon analogue à ce que l'on a présenté pour la sauvegarde de points de reprise (voir section 4.4.1.2.1).

En ce qui concerne $t_{CopyPrivate}(p, V_{data})$, nous le calibrons de façon analogue à la

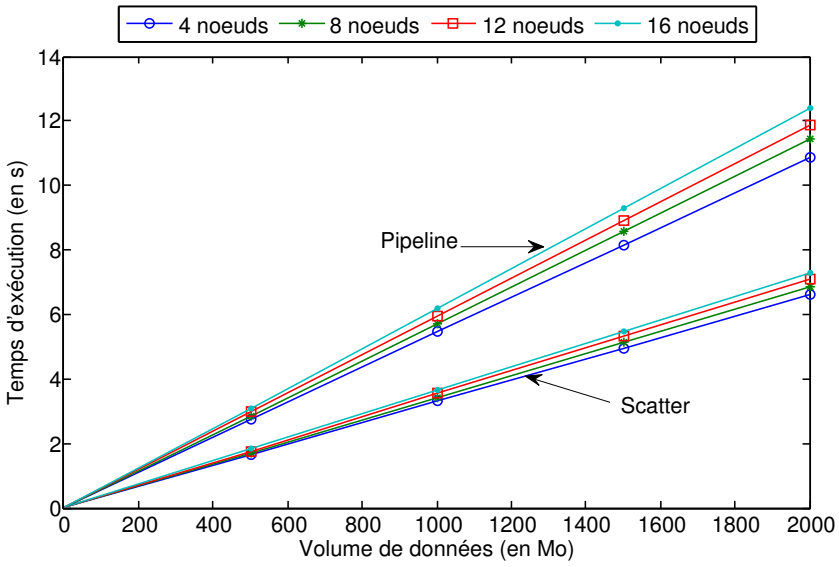

FIGURE 4.11 – Calibration de $t_{Scatter}(N, 1)$ et $t_{Pipeline}(N, 1)$

sauvegarde de points de reprise (voir section 4.4.1.2.1) mais avec la mémoire RAM comme support de stockage. Pour différents nombres (1, 4, 6, 9, 12) de processus localisés sur le même noeud, nous mesurons $t_{CopyPrivate}(p, V_{data})$ pour une donnée de taille variable (0 Mo, 500 Mo, 1000 Mo, 1500 Mo, 2000 Mo).

Sur la figure 4.12, nous présentons les résultats de calibration pour $t_{CopyPrivate}(p, V_{data})$ $t_{CopyPrivate}(1, V_{data})$ est égal à $t_{enregistrement}(V_{data})$ pour un enregistrement de message sur mémoire RAM.

Nous avons besoin de prendre en compte $t_{CopyPrivate}(p, V_{data})$ lorsque nous considérons plusieurs processus par noeud avec des algorithmes hybrides de diffusion de données étant donné que ce temps n'est pas négligeable. De plus, pour un volume de données fixé, nous remarquons que $t_{CopyPrivate}(p, V_{data})$ demeure quasiment inchangé lorsque nous considérons un nombre croissant de processus par noeud. Ceci s'explique par le fait que la donnée partagée est simultanément enregistrée sans congestion dans la mémoire RAM du noeud.

4.4.2 Précision des estimations

Dans cette section, nous cherchons à comparer la consommation énergétique obtenue par notre estimateur une fois que la calibration est faite (mais avant d'exécuter l'application) à la consommation énergétique mesurée réellement par les wattmètres OMEGAWATT pendant l'exécution de l'application.

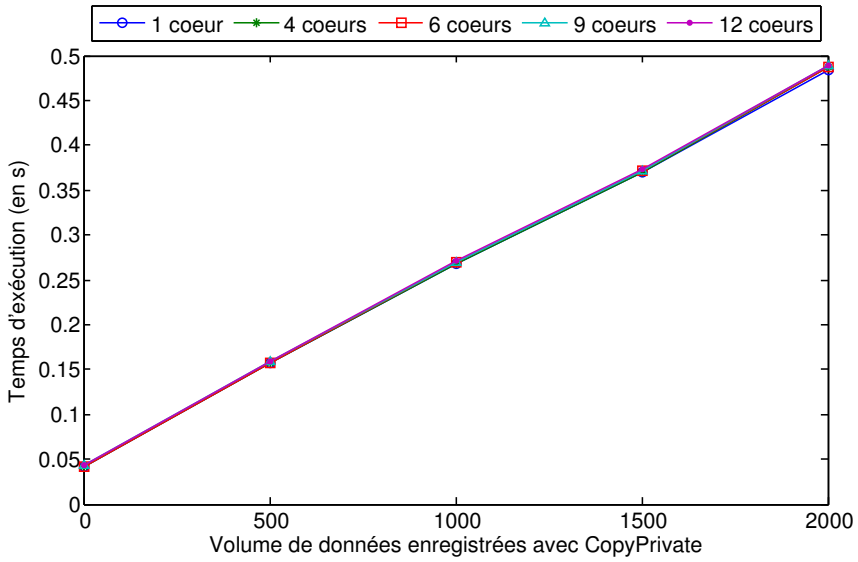

FIGURE 4.12 – Calibration de $t^{Noeud_i}_{CopyPrivate}(p)$

4.4.2.1 Cas de la tolérance aux pannes

Nous considérons 4 applications de calcul haute performance : CM1 [1] avec une résolution 2400x2400x40 exécuté sur 144 processus de la grappe *Taurus* et 3 NAS [2] en classe D (SP, BT, et EP) exécutés sur 144 processus (*i.e.,* 12 noeuds de 12 coeurs par noeud) de la grappe *Taurus*.

Grâce à l'infrastructure de wattmètres externes OMEGAWATT, nous mesurons pour chaque application la consommation énergétique pendant l'exécution de l'application avec et sans l'activation du service de tolérance aux pannes. Plus précisément, nous avons instrumenté le code source des implémentations des différents protocoles de tolérance aux pannes pour activer ou non chacune des opérations décrites précédemment : sauvegarde de points de reprise, enregistrement de messages (sur mémoire RAM ou disque dur local) et coordination. Ainsi, nous obtenons la consommation énergétique réelle pour chaque opération. Chaque mesure énergétique est réalisée 30 fois et nous considérerons par la suite les valeurs moyennes.

Nous avons estimé et mesuré le coût énergétique de tous les messages enregistrés dans le cadre du protocole non coordonné. Nous avons estimé et mesuré le coût énergétique d'une seule sauvegarde de point de reprise et donc d'une seule coordination. Pour pouvoir mesurer la consommation énergétique d'une seule sauvegarde de point de reprise, nous avons utilisé un intervalle entre deux sauvegardes de points de reprise plus grand que la moitié de la durée de l'application de calcul haute performance. Ainsi, la première et unique sauvegarde de points de reprise aura lieu dans

1. Cloud Model 1 : http://www.mmm.ucar.edu/people/bryan/cm1/
2. NAS : http://www.nas.nasa.gov/publications/npb.html

la deuxième moitié de l'exécution de l'application.

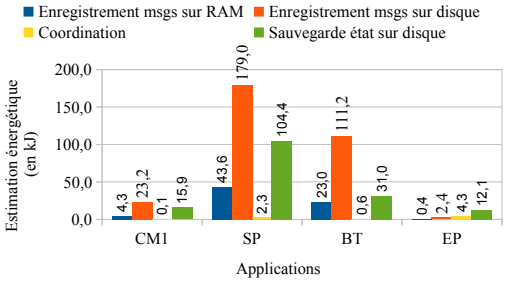

FIGURE 4.13 – Estimations des consommations énergétiques (en kJ) des opérations liées à la tolérance aux pannes

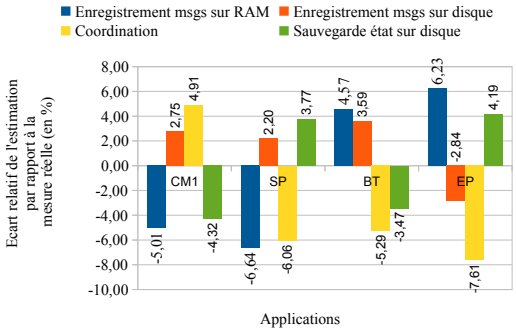

FIGURE 4.14 – Écarts relatifs (en %) entre les consommations énergétiques estimées et mesurées des opérations identifiées dans les protocoles de tolérance aux pannes

Sur la figure 4.13, nous affichons les estimations énergétiques pour les différentes opérations identifiées dans les protocoles de tolérance aux pannes. Sur la figure 4.14, nous affichons les écarts relatifs (en pourcentage) entre la consommation énergétique estimée et celle mesurée réellement. La figure 4.14 montre que les estimations énergétiques fournies sur la figure 4.13 sont précises. En effet, les écarts relatifs entre les consommations énergétiques estimées et mesurées sont faibles. La plus mauvaise estimation affiche un écart de 7.6% par rapport à la valeur mesurée pour la coordination avec EP. L'écart moyen sur l'ensemble des tests est de 4.9 %.

Nous constatons que nous estimons un peu moins bien la coordination par rapport à l'enregistrement de messages ou à la sauvegarde de points de reprise. Ceci est lié au fait que cette opération dure beaucoup moins longtemps que l'enregistrement de tous les messages. Ceci est également dû au fait que cette opération est évaluée à partir de l'estimation deux sous-opérations ($t_{attente}$ et $t_{synchro}$) ce qui génére plus d'imprécisions dans notre estimation.

Nous montrerons dans le chapitre suivant comment de telles estimations énergétiques peuvent permettre de réduire la consommation énergétique liée aux protocoles de tolérance aux pannes si elles sont connues avant de pré-exécuter l'application.

4.4.2.2 Cas de la diffusion de données

Nous considérons quatre classes d'applications de diffusion de données mettant en jeu différents contextes d'exécution présentés dans le tableau 4.4.2.2.

Pour les quatre scénarios d'exécution, nous choisissons un volume total assez grand pour tous les messages diffusés afin chaque scénario d'exécution dure plusieurs secondes et que les mesures d'énergie soient ainsi possibles à l'aide des wattmètres OMEGAWATT. Les paramètres de la taille du message diffusé, du nombre de noeuds et du nombre de processus par noeud sont choisis de telle sorte :

Nom	Nombre de messages	Taille de chaque message	Nombre de noeuds	Processus par noeud
A	2000	1 Mo	14	8
B	80	25 Mo	16	1
C	4	500 Mo	10	4
D	1	1.75 Go	6	12

TABLE 4.1 – Contextes d'exécution considérés pour les 4 applications de diffusion de données

– qu'un seul paramètre n'ait pas été mesuré (applications B et D) ;
– que deux paramètres n'aient pas été mesurés (application C) ;
– que trois paramètres n'aient pas été mesurés (application A).

Pour chaque application (A, B, C, D) et pour chaque algorithme de diffusion de données (*MPI/SAG*, *MPI/Pipeline*, *Hybrid/SAG*, *Hybrid/Pipeline*), nous estimons la consommation énergétique en additionnant les énergies consommées par les différentes opérations identifiées dans chaque algorithme. Nous estimons l'énergie de chaque opération en multipliant l'énergie liée à la diffusion d'un seul message par le nombre de messages de l'application (A, B, C ou D).

D'autre part, nous mesurons la consommation énergétique correspondante à chaque algorithme et à chaque application. Dans nos expériences, l'opération *Pipeline* est réalisée avec une taille de pièce fixée à 128 Ko dans l'implémentation de l'algorithme. Chaque mesure énergétique est faite 30 fois et nous considérerons par la suite les valeurs moyennes.

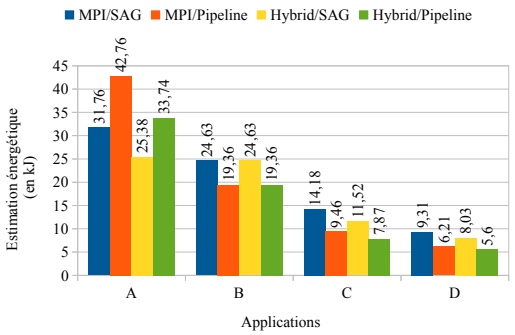

FIGURE 4.15 – Estimations des consommations énergétiques (en kJ) des quatre algorithmes pour les quatre applications

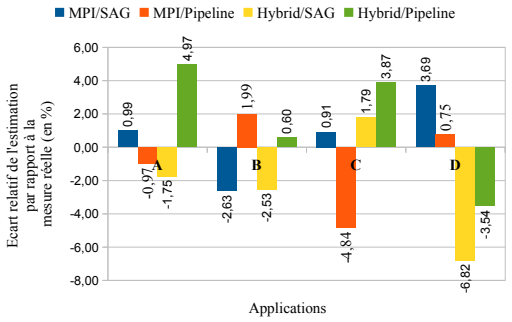

FIGURE 4.16 – Écarts relatifs (en %) entre les consommations énergétiques mesurées et estimées pour les quatre algorithmes de diffusion de données

Sur la figure 4.15, nous affichons les estimations énergétiques pour les différents algorithmes de diffusion de données. Sur la figure 4.16, nous affichons les écarts relatifs (en pourcentage) entre la consommation énergétique estimée et celle mesurée réellement. Pour rapporter l'estimation énergétique au coût énergétique d'un seul message de taille V_{data}, il suffit de diviser l'estimation affichée pour tous les messages

de l'application (A, B ou C) par le nombre de messages diffusés. La figure 4.16 montre que les estimations énergétiques fournies sur la figure 4.15 sont précises. En effet, les écarts relatifs entre les consommations énergétiques estimées et mesurées sont faibles. La plus mauvaise estimation affiche un écart de -6.82 % par rapport à la valeur mesurée pour l'algorithme *Hybrid/SAG* avec l'application D. Nous montrerons dans le chapitre suivant comment de telles estimations énergétiques peuvent permettre de réduire la consommation énergétique de la diffusion de données si elles sont connues à l'avance.

4.5 Conclusions du chapitre

Dans ce chapitre, nous avons présenté notre approche pour estimer avec précision la consommation énergétique d'un service applicatif donné. Nous avons notamment appliqué notre approche aux protocoles de tolérance aux pannes et des algorithmes de diffusion de données. En ce qui concerne la tolérance aux pannes, nous nous sommes focalisés sur la phase de fonctionnement sans pannes. Nous avons considéré le cas des protocoles coordonnés et celui des protocoles non coordonnés. En ce qui concerne la diffusion de données, nous avons considéré deux algorithmes utilisés dans MPI (*MPI/SAG* et *MPI/Pipeline*) et deux algorithmes hybrides combinant chacun des deux derniers algorithmes avec OpenMP (respectivement *Hybrid/SAG* et *Hybrid/Pipeline*).

Notre approche d'estimation consiste d'abord à identifier les opérations que nous retrouvons dans les différents protocoles ou algorithmes du service applicatif étudié. Ensuite, afin d'adapter nos modèles théoriques aux spécificités de la plate-forme considérée, nous procédons à une calibration énergétique qui consiste à recueillir un ensemble de mesures de la puissance électrique et des temps d'exécution pour chacune des opérations identifiées. Pour calibrer la plate-forme, le calibrateur recueille les paramètres décrivant la plate-forme d'exécution, tels que le nombre de noeuds ou le nombre de coeurs par noeud. Grâce à une telle calibration, les estimations énergétiques que nous fournissons peuvent s'adapter à n'importe quelle plate-forme. Une fois que la calibration est achevée, l'estimateur s'appuie sur les résultats de calibration ainsi que sur une description de contexte d'exécution qui consiste à fournir un ensemble de paramètres tels que le nombre de noeuds mis en jeu ou le volume de données à diffuser.

Nous avons montré dans ce chapitre que nos estimations énergétiques sont précises pour chacune des opérations que ce soit pour la tolérance aux pannes ou pour la diffusion de données. En effet, en comparant les estimations énergétiques que nous sommes capables de fournir aux mesures énergétiques de chacune des opérations, nous avons pu montré que les écarts relatifs étaient faibles. En ce qui concerne la tolérance aux pannes, les écarts relatifs entre les estimations et les mesures énergétiques sont égaux à 4.9 % en moyenne et n'excédent pas 7.6 %. En ce qui concerne la diffusion de données, les écarts relatifs n'excédent pas 6.82 % pour les différents contextes d'exécution considérés. Dans le chapitre suivant, nous montrerons com-

ment il est possible d'utiliser nos estimations énergétiques pour réduire la consommation énergétique lors de l'exécution de services applicatifs pour le calcul très haute performance.

5 Consommer moins d'énergie dans les infrastructures de calcul très haute performance

Dans ce chapitre, nous proposons différentes solutions pour consommer moins d'énergie en s'appuyant principalement sur les résultats que nous avons mis en évidence dans les chapitres 3 et 4. D'une part, nous pouvons réduire la consommation des applications de calcul très haute performance en s'appuyant sur les estimations des différentes versions d'un service applicatif que nous sommes capables de fournir, pour choisir la version qui consomme le moins d'énergie en fonction du contexte d'exécution. D'autre part, une solution pour réduire la consommation énergétique d'une infrastructure de calcul très haute performance est d'exploiter l'hétérogénéité électrique des noeuds de calcul en choisissant les noeuds qui ont la plus basse puissance électrique au repos.

La section 5.1 montre comment nous exploitons les estimations énergétiques des différents protocoles de tolérance aux pannes [Diouri et al., 2013e, Diouri et al., 2013h] et des différents algorithmes de diffusion de données [Diouri et al., 2013g] pour réduire la consommation énergétique de ces deux services applicatifs. Dans la section 5.2, nous montrons comment il est possible de réduire la consommation d'énergie en exploitant l'hétérogénéité de la puissance électrique qui caractérise des noeuds identiques issus d'une même grappe de calcul [Diouri et al., 2013d].

5.1 Choix du service applicatif en exploitant les estimations énergétiques

Pour un même service applicatif, plusieurs implémentations sont possibles. En fonction de l'application considérée, la version du service qui consomme le moins d'énergie peut changer d'une application à l'autre. Notre estimation permet de prévoir cela et d'interagir avec l'utilisateur afin de choisir la version du service en toute connaissance de cause. Par ce choix, l'utilisateur peut donc réduire la consommation énergétique du service considéré. Afin de mettre en exergue cela, nous étudions les cas de la tolérance aux pannes dans la section 5.1.1 et de la diffusion de données dans la section 5.1.2.

5.1.1 Tolérance aux pannes

Les deux familles de protocoles étudiés pour la tolérance aux pannes sont les protocoles coordonnés et non coordonnés. Nous considérons les 4 applications de calcul haute performance que nous avons étudiées dans la section 4.4.2.1 : CM1 avec une résolution 2400x2400x40 et 3 NAS [1] en classe D (SP, BT, et EP) s'exécutant sur 144 processus (*i.e.,* 12 noeuds exécutant chacun 12 processus, 1 par coeur).

Pour chaque application et pour chaque protocole de tolérance aux pannes, nous estimons la consommation énergétique en nous appuyant sur les différentes opérations identifiées (voir chapitre 4). D'abord, nous mettons en évidence que le coût énergétique d'une opération liée à la tolérance aux pannes est très dépendant de l'application. Ensuite, nous montrons comment les estimations énergétiques des différentes opérations identifiées dans 4.1.1 peuvent guider l'utilisateur dans son choix du protocole de tolérance aux pannes qui consomme le moins d'énergie.

Sur la figure 5.1, nous affichons la consommation énergétique évaluée par notre estimateur pour chaque opération et pour chaque application de calcul haute performance considérée. Comme nous l'avons expliqué dans la section 4.4.2.1, nous fournissons les estimations énergétiques d'une seule sauvegarde de points de reprise et donc d'une seule coordination.

La figure 5.1 montre que la consommation énergétique des opérations ne sont pas les mêmes d'une application à l'autre. Par exemple, la consommation énergétique de l'enregistrement de messages sur RAM avec l'application SP est plus de 10 fois plus important que celui avec l'application CM1. Ceci est dû au fait que CM1 échange beaucoup moins de messages que l'application SP.

Nous pouvons obtenir la consommation énergétique totale de chaque protocole de tolérance aux pannes en additionnant les consommations énergétiques des opérations considérées dans chaque protocole. La sauvegarde de points de reprise est une opération utilisée dans les deux protocoles de tolérance aux pannes : la sauvegarde coordonnée et celle non coordonnée. Pour la sauvegarde non coordonnée de points de reprise, nous y ajoutons la consommation énergétique de l'enregistrement

1. NAS : http://www.nas.nasa.gov/publications/npb.html

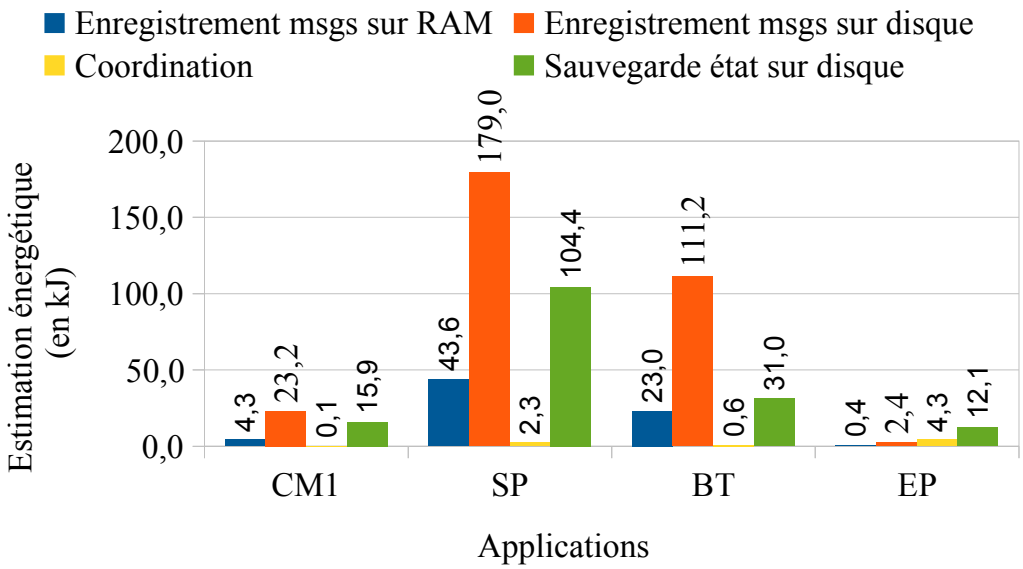

FIGURE 5.1 – Estimations des consommations énergétiques (en kJ) des opérations liées à la tolérance aux pannes (Figure 4.13)

de messages. Pour la sauvegarde coordonnée, nous additionnons la consommation énergétique de la sauvegarde de points de reprise et celle des coordinations.

La sauvegarde de points de reprise est ainsi combinée soit à l'enregistrement de messages dans le protocole coordonné, soit à la coordination dans le protocole coordonné. Ainsi, pour comparer ces deux protocoles d'un point de vue énergétique, nous devons de comparer le coût énergétique des coordinations et celui de l'enregistrement de messages. Dans nos expériences, nous avons considéré un enregistrement de messages soit en mémoire RAM soit sur le disque dur local. La coordination consommera d'autant plus qu'il y aura des messages volumineux qui restent en cours de transmission aux moments de la synchronisation des processus. L'enregistrement de messages consommera d'autant plus que le nombre et la taille des messages échangés lors de l'exécution de l'application sont importants.

La figure 5.1 montre que d'une application à l'autre, le protocole qui consomme le moins d'énergie n'est pas toujours le même. En général, déterminer le protocole de tolérance qui consomme le moins d'énergie consiste en un compromis entre le volume de données échangées et le coût de la coordination. Pour les applications CM1, SP et BT, le protocole qui consomme le moins d'énergie est le protocole coordonné car les valeurs énergétiques pour l'enregistrement de messages (aussi bien RAM que sur disque dur) sont supérieures aux valeurs énergétiques de la coordination (voir figure 5.1). En effet, le volume de données à enregistrer pour ces applications est relativement important et génère une consommation énergétique plus élevée. À l'opposé, le protocole de tolérance aux pannes qui consomme le moins d'énergie pour l'application EP est le protocole non coordonné.

Ces conclusions sont spécifiques au cas où il n'y a qu'une seule sauvegarde de points de reprise et donc une seule coordination lors de l'exécution de ces applications. Si l'utilisateur souhaite plus de fiabilité et ce particulièrement pour les applications qui durent longtemps (plusieurs heures), il devra choisir un intervalle de points de reprise réduit et donc un nombre plus élevé de sauvegardes de points de reprise et de coordinations. Cet intervalle de points de reprise peut influencer le choix du protocole qui consomme le moins d'énergie. En effet, si par exemple, lors de l'exécution de SP, il y a plus de 19 sauvegardes de points de reprise soit 19 coordinations, le coût énergétique lié aux coordinations sera plus élevé. Par conséquent, il sera préférable d'utiliser le protocole non coordonné pour réduire la consommation énergétique liée à la tolérance aux pannes.

L'intervalle de points de reprise peut être choisi en s'appuyant sur les modèles qui définissent pour l'intervalle optimal : celui qui permet de maximiser la fiabilité tout en minimisant les pertes en termes performances [Young, 1974, Daly, 2006].

Dans le cas où on utiliserait plus de processus pour l'exécution d'une même application, le coût énergétique lié aux coordinations sera plus importante. Toutefois, le coût énergétique de l'enregistrement de messages risque aussi de croître puisque l'augmentation du nombre de processus entraînera plus de communications entre les processus. Il y aura donc plus de messages à échanger et donc plus de messages à enregistrer.

Ainsi, en fournissant de telles estimations énergétiques avant l'exécution d'une application de calcul très haute performance, nous permettons à l'utilisateur de choisir le protocole de tolérance aux pannes qui consomme le moins d'énergie en fonction du nombre de sauvegardes de points de reprise qu'il souhaite réaliser durant l'exécution de son application.

5.1.2 Diffusion des données

Les quatre algorithmes de diffusion de données étudiés sont *MPI/SAG*, *MPI/-Pipeline*, *Hybrid/SAG* et *Hybrid/Pipeline*. Notre but est de comparer ces quatre algorithmes d'un point de vue énergétique en s'appuyant sur les classes d'applications de diffusion de données que nous avons étudiées dans la section 4.4.2.2. Elles sont à nouveau présentées ci-dessous :

Nom	Nombre de messages	Taille de chaque message	Nombre de noeuds	Processus par noeud
A	2000	1 Mo	14	8
B	80	25 Mo	16	1
C	4	500 Mo	10	4
D	1	1.75 Go	6	12

Pour chaque application (A, B, C, D) et pour chaque algorithme de diffusion de données (*MPI/SAG*, *MPI/Pipeline*, *Hybrid/SAG* et *Hybrid/Pipeline*), nous estimons la consommation énergétique en additionnant les énergies consommées par

les différentes opérations identifiées dans chaque algorithme. Dans nos expériences, l'opération *Pipeline* est réalisée avec une taille de paquet fixée à 128 Ko.

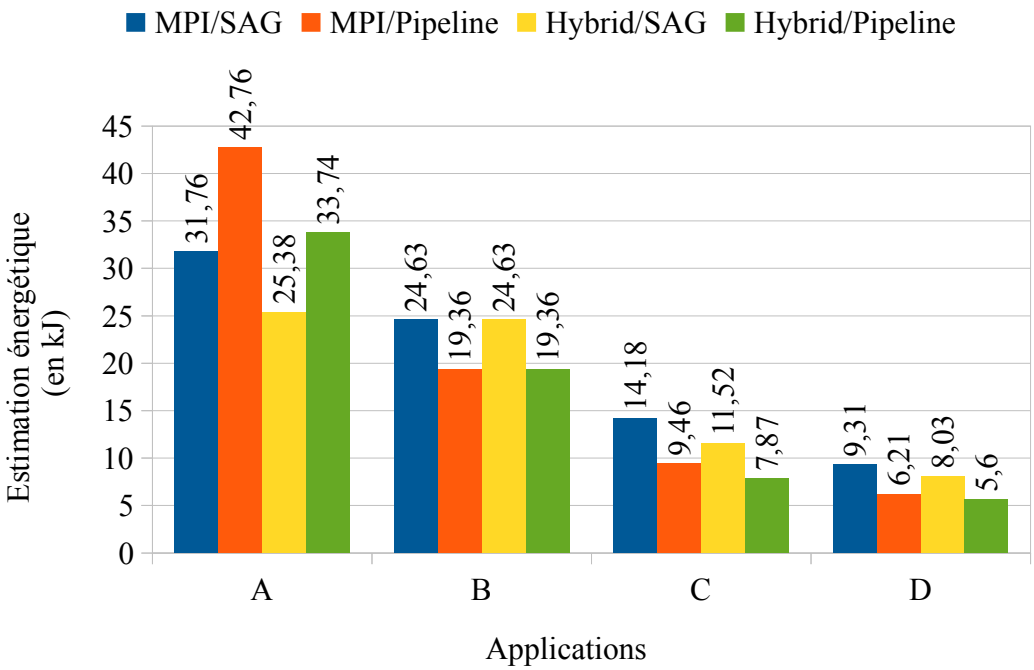

FIGURE 5.2 – Estimations des consommations énergétiques (en kJ) des quatre algorithmes pour les quatre applications (Figure 4.15)

Sur la figure 5.2, nous affichons la consommation énergétique évaluée à l'aide de l'estimateur pour chaque algorithme de diffusion et pour chaque application. La figure 5.2 montre que d'une application à l'autre, l'algorithme de diffusion de données qui consomme le moins d'énergie n'est pas toujours le même. Dans un premier temps, nous remarquons que chaque algorithme hybride consomme moins d'énergie que son homologue pur MPI, et ce en particulier lorsque nous considérons plusieurs processus par noeud (applications A, C et D).

En général, déterminer l'algorithme qui consomme le moins d'énergie dépend d'un compromis entre le volume de données diffusées et le nombre de noeuds mis en jeu. Pour l'application A, l'algorithme qui consomme le moins d'énergie est *Hybrid/SAG* étant donné que le nombre de paquets qui sont transmis en pipeline est relativement petit du fait que le volume de données à diffuser est faible. À l'opposé, l'algorithme de diffusion de données qui consomme le moins d'énergie pour les applications B, C et D est *Hybrid/Pipeline*, étant donné que le nombre de paquets transmis en pipeline devient suffisamment important par rapport au nombre de processus impliqués dans la diffusion de données.

Ainsi, en fournissant de tels estimations de consommation énergétiques avant l'exécution de l'application, nous pouvons permettre à l'utilisateur de choisir le

meilleur algorithme de diffusion de données en termes de consommation énergétique. On pourrait penser que la consommation énergétique d'un algorithme est complètement liée à sa performance (*i.e.,* son temps d'exécution). Nos résultats d'estimation montrent que ceci n'est pas vrai. En effet, bien que la figure 5.2 montre que les consommations énergétiques des algorithmes hybrides sont plus basses que celles des algorithmes utilisés dans MPI dans les applications A, C et D, les temps d'exécution estimés des algorithmes hybrides sont légèrement plus élevés que les temps d'exécution des algorithmes utilisés dans MPI. En effet, la puissance électrique des algorithmes hybrides pendant les opérations MPI (*Scatter*, *AllGather* et *Pipeline*) est beaucoup plus basse que celle des algorithmes utilisés dans MPI. Ceci est dû au fait que dans les algorithmes hybrides, seul un processus est actif lors du *Scatter*, du *AllGather* et du *Pipeline* alors que dans les algorithmes utilisés dans MPI, tous les processus sont actifs pendant les opérations MPI (voir la figure 4.5).

5.2 Choix des noeuds de calcul en exploitant leur hétérogénéité électrique

Cette section présente FLIP (*First Less Idle Power*), une approche efficace en énergie qui tire profit des différences de puissance électrique au repos des noeuds issus d'une même grappe homogène afin de faire des économies d'énergie durant l'exécution d'applications sur des systèmes distribuées à large échelle, et cela sans perdre en performance.

5.2.1 Méthodologie

Le choix des noeuds à attribuer lors des demandes de réservations de ressources de calcul est un des critères qu'il faut prendre en compte dans les algorithmes d'ordonnancement. Les approches irréalistes d'attribution de noeuds lors des demandes de réservations de ressources supposent que tous les noeuds d'une même grappe ont la même puissance électrique. En effet, dans [Ge et al., 2009, Steinder et al., 2008, Kim et al., 2007], les auteurs considèrent que la consommation énergétique de N noeuds d'une même grappe est égale à la consommation énergétique d'un noeud quelconque multipliée par N. Afin d'étudier les différences entre une affectation de noeuds basée sur la puissance électrique des noeuds et une approche irréaliste d'attribution de noeuds où tous les noeuds sont supposés avoir la même puissance électrique, nous distinguons trois politiques d'attribution de noeuds reposant sur des hypothèses irréalistes :

- *Homogène_MIN* : tous les noeuds ont la même puissance électrique que le noeud de la grappe qui consomme le moins. Il s'agit du cas irréel idéal.
- *Homogène_MEDIANE* : tous les noeuds ont la même puissance électrique qu'un noeud de la grappe qui aurait une puissance électrique égale à la médiane des puissances électriques des noeuds de la grappe. Il s'agit du scénario irréel qui est le plus représentatif de la réalité.

– *Homogène_MAX* : tous les noeuds ont la même puissance électrique que le noeud de la grappe qui consomme le plus. Il s'agit du pire cas irréel.

Nous avons montré dans la section 3.3.1 que la réalité est différente. C'est la raison pour laquelle nous proposons maintenant des approches réalistes d'attribution de noeuds qui ne supposent pas que la consommation énergétique est égale pour tous les noeuds identiques d'une même grappe. Ainsi, nous considérons que la consommation énergétique de toute la grappe est égale à la somme des énergies mesurées pour chaque noeud. Nous distinguons trois politiques réalistes d'attribution de noeuds :

– *IDsTries* : les noeuds sont affectés à l'utilisateur dans l'ordre croissant des identifiants des noeuds dans la grappe. C'est la politique par défaut de notre plate-forme d'exécution.
– *FLIP* : les noeuds sont affectés à l'utilisateur dans l'ordre croissant des consommations électriques au repos de chaque noeud. C'est l'approche que nous proposons dans ce chapitre : nous affectons en priorité à l'utilisateur les noeuds dont la puissance électrique au repos est la plus basse.
– *FMIP* : les noeuds sont affectés à l'utilisateur dans l'ordre décroissant des consommations électriques au repos par noeud. C'est le pire scénario réaliste.

Afin de comparer ces politiques d'attribution de noeuds, nous mesurons d'abord la puissance électrique moyenne au repos pour chaque noeud noeud de chaque grappe de calcul considérée. En ce qui concerne les politiques irréalistes d'attribution de noeuds, nous considérons dans nos simulations que la puissance électrique mesurée est la même pour tous les noeuds qui exécutent la même application. Quant aux approches d'attribution de noeuds réalistes, nous trions les noeuds dans des listes ordonnées qui correspondent aux politiques d'attribution de noeuds décrites précédemment.

Pour effectuer nos simulations d'attribution de noeuds, nous avons considéré les grappes de calcul *Sagittaire*, *Stremi* et *Taurus* décrites dans le tableau 3.1, et sur lesquelles nous avons mesuré la puissance électrique au repos de tous les noeuds afin d'obtenir les listes ordonnées des noeuds associées à chacune des politiques d'attribution de noeuds. La figure 5.3 montre la puissance électrique des listes ordonnées des noeuds des trois grappes de calcul selon la politique d'attribution de noeuds considérée. À chaque fois qu'un utilisateur souhaite réserver un certain nombre de noeuds ou d'exécuter une application sur un certain nombre de machines, une politique d'attribution de noeuds donnée lui affecte en priorité les premiers noeuds disponibles de sa liste ordonnée de noeuds.

Du point de vue de l'utilisateur, son application consommera moins d'énergie s'il choisit de l'exécuter sur les noeuds obtenus grâce à *FLIP*, c'est-à-dire sur les noeuds qui consomment le moins. Ainsi, s'il est facturé en fonction de sa consommation énergétique, *FLIP* lui permettra d'économiser de l'énergie et donc de l'argent. S'il est facturé en fonction du nombre de noeuds, il n'y aura aucune incidence sur sa facture s'il choisit d'utiliser des noeuds plutôt que d'autres. Par ailleurs, s'il utilise toute la plate-forme, il aura forcément tous les noeuds et ne pourra donc pas faire un choix. Ainsi, pour que *FLIP* soit financièrement profitable à l'utilisateur, il faut

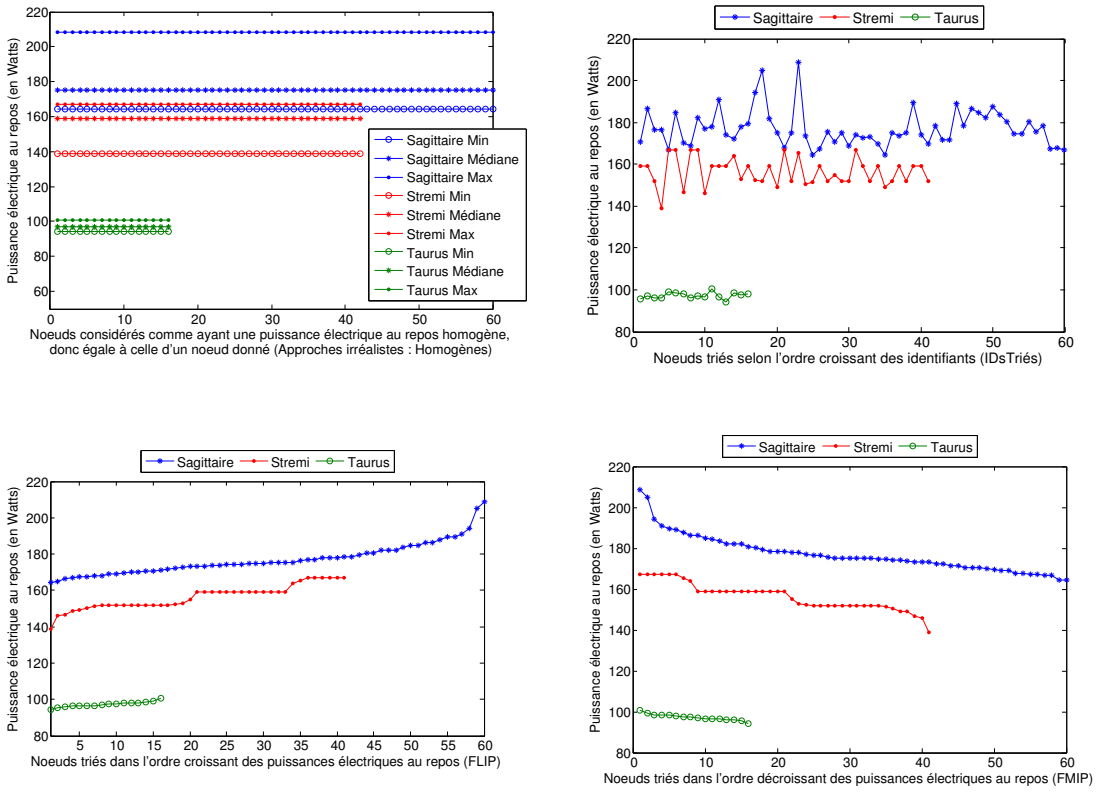

FIGURE 5.3 – Listes ordonnées des noeuds permettant l'application des différentes politiques d'attribution de noeuds (Homogènes, IDsTries, FLIP, FMIP)

qu'il n'utilise qu'une partie de la plate-forme et qu'il soit facturé en fonction de sa consommation énergétique. Nous évaluons ces gains énergétiques dans la section 5.2.2.

Pour que le propriétaire de l'infrastructure économise de l'énergie en utilisant *FLIP* plutôt qu'une autre approche, il est indispensable qu'il éteigne les noeuds qui restent inutilisés. En effet, si tous les noeuds de l'infrastructure restent allumés, la puissance électrique totale de la plate-forme ne baissera pas. Étant donné que les noeuds qu'il peut éteindre sont les noeuds inutilisés, il a tout intérêt à s'arranger pour que ces noeuds inutilisés soient ceux qui consomment le plus. *FLIP* est dans cette perspective dans la mesure où elle permet d'utiliser en priorité les noeuds qui consomment le moins. Pour que *FLIP* soit financièrement profitable au propriétaire de l'infrastructure, il faut que celui-ci facture l'utilisateur en fonction du nombre de noeuds (et non en fonction de la consommation énergétique). En effet, s'il facture l'utilisateur en fonction de la consommation énergétique, la facture d'électricité du propriétaire sera tout de même plus réduite en éteignant les noeuds qui consomment le plus mais sa recette sera moins importante en attribuant à l'utilisateur les noeuds qui consomment le moins. Dans la section 5.2.3, nous évaluons la puissance électrique économisée du point de vue du propriétaire de l'infrastructure, dans le cas où des noeuds restent inutilisés et qu'il décide de les éteindre. Pour cela, nous simulons donc l'extinction des noeuds inutilisés, en considérant que leur consommation énergétique est nulle lorsqu'ils sont éteints.

5.2.2 Évaluation du gain énergétique lors des exécutions d'applications de calcul haute performance

Dans cette section, nous cherchons à évaluer le gain énergétique du point de vue de l'utilisateur lorsqu'il choisit d'utiliser les noeuds sur lesquels il souhaite d'exécuter des applications. Comme annoncé dans la section 5.2.1, ceci n'est valable que s'il est facturé en fonction de la consommation énergétique et s'il n'utilise qu'une partie de la plate-forme de calcul haute performance

Pour évaluer ce gain énergétique, nous considérons quatre applications de calcul haute performance, quatre NAS [2] (LU, CG, EP, MG) s'exécutant en classe C sur :

- 8, 16, 32 et 64 coeurs de *Sagittaire* ;
- 32, 64, 128 et 256 coeurs de *Stremi* ;
- 8, 16, 32 et 64 coeurs de *Taurus*.

Nous utilisons deux coeurs par noeud pour *Sagittaire* ; seize coeurs par noeud pour *Stremi* ; et huit coeurs par noeud pour *Taurus*. Les nombres de noeuds et les nombres de coeurs par noeud ont été choisis afin d'utiliser plus de 33 % mais moins de 100 % des noeuds de chaque grappe. Nous n'utilisons qu'une partie de chaque grappe, car si celle-ci était utilisé à 100 %, la puissance électrique globale serait la même quelle que soit la politique d'attribution de noeuds.

Nous comparons l'énergie consommée par les quatre applications de calcul haute

2. NAS : http://www.nas.nasa.gov/publications/npb.html

performance ordonnancées sur les noeuds attribués par l'approche *FLIP* par rapport aux noeuds affectés par les autres politiques d'attribution de noeuds. Nous rappelons que dans le cas des politiques *Homogène_MIN*, *Homogène_MEDIANE* et *Homogène_MAX*, l'énergie de tous les noeuds est évaluée en mesurant un seul noeud particulier (respectivement celui qui consomme le moins, le médian ou celui qui consomme le plus) et en multipliant cette mesure par le nombre de noeuds comme si la consommation énergétique était homogène pour tous les noeuds identiques d'une grappe. Chaque mesure énergétique est faite 25 fois ; nous calculons la valeur moyenne sur les 25 mesures. La figure 5.4 montre pour les trois grappes considérées la proportion d'énergie économisée avec *FLIP* en comparaison aux autres politiques d'attribution de noeuds. Les axes des ordonnées des trois graphiques correspondant aux trois grappes sont mis à la même échelle. Chaque point d'une courbe représente la proportion d'énergie économisée en moyenne sur les quatre applications de calcul haute performance. La barre autour de chaque point représente l'intervalle où se situent les proportions des énergies économisées sur chaque application.

Sur la figure 5.4, les courbes ne sont pas superposables pour chacune des trois grappes. Cela confirme l'hétérogénéité électrique qui existe sur chacune des trois grappes. Cette hétérogénéité électrique est plus ou moins importante en fonction de la grappe considérée. Plus l'hétérogénéité électrique est importante, plus l'énergie économisée grâce à la politique d'attribution de noeuds *FLIP* est importante. En effet, comme nous pouvons le voir dans la figure 5.4, relativement à *FMIP*, *FLIP* peut économiser jusqu'à 17 % d'énergie sur la grappe *Sagittaire*, jusqu'à 11% sur la grappe *Stremi* et moins de 5% sur *Taurus*.

Par rapport à *IDsTries*, nous économisons avec *FLIP* jusqu'à 5% sur la grappe *Sagittaire*, jusqu'à 8% sur *Stremi* et moins de 2% sur *Taurus*. Ceci est visible lorsque la proportion de noeuds utilisés est de $4/60$ pour *Sagittaire*, $2/42$ pour *Stremi* et $1/16$ pour *Taurus*. Même si la grappe *Sagittaire* est plus hétérogène électriquement que *Stremi*, la proportion d'énergie économisée semble être plus importante avec *Stremi* par rapport à l'approche *IDsTries*. Ceci est principalement dû au fait que les deux premiers noeuds de la liste ordonnée obtenue pour *Stremi* sont parmi les noeuds les plus consommateurs tandis que les quatre premiers noeuds de la liste ordonnée obtenue pour *Sagittaire* sont parmi les noeuds dont la puissance électrique se situe au niveau de la médiane.

De plus, la figure 5.4 montre que les courbes décroissent plus la proportion des noeuds utilisés augmente. Plus nous utilisons des noeuds, moins la proportion d'énergie économisée est importante. En effet, plus nous utilisons des noeuds, plus l'hétérogénéité électrique diminue en moyenne. Si nous utilisons tous les noeuds d'une grappe, la consommation énergétique est la même, quelle que soit la politique d'attribution de noeuds.

Par ailleurs, nous observons que les écarts par rapport aux valeurs moyennes sur les quatre applications de calcul haute performance sont négligeables spécialement pour *Sagittaire* et *Taurus*. Même si le temps d'exécution est différent d'une application à l'autre, cette observation confirme que la proportion d'énergie économisée

dépend très peu de la charge de travail qui s'exécute sur les noeuds utilisés. Ces écarts semblent être plus importants pour la grappe *Stremi* sûrement à cause du manque de précision des mesures prises avec les unités de distribution de l'énergie (PDUs) supervisant la consommation énergétique de la grappe *Stremi*.

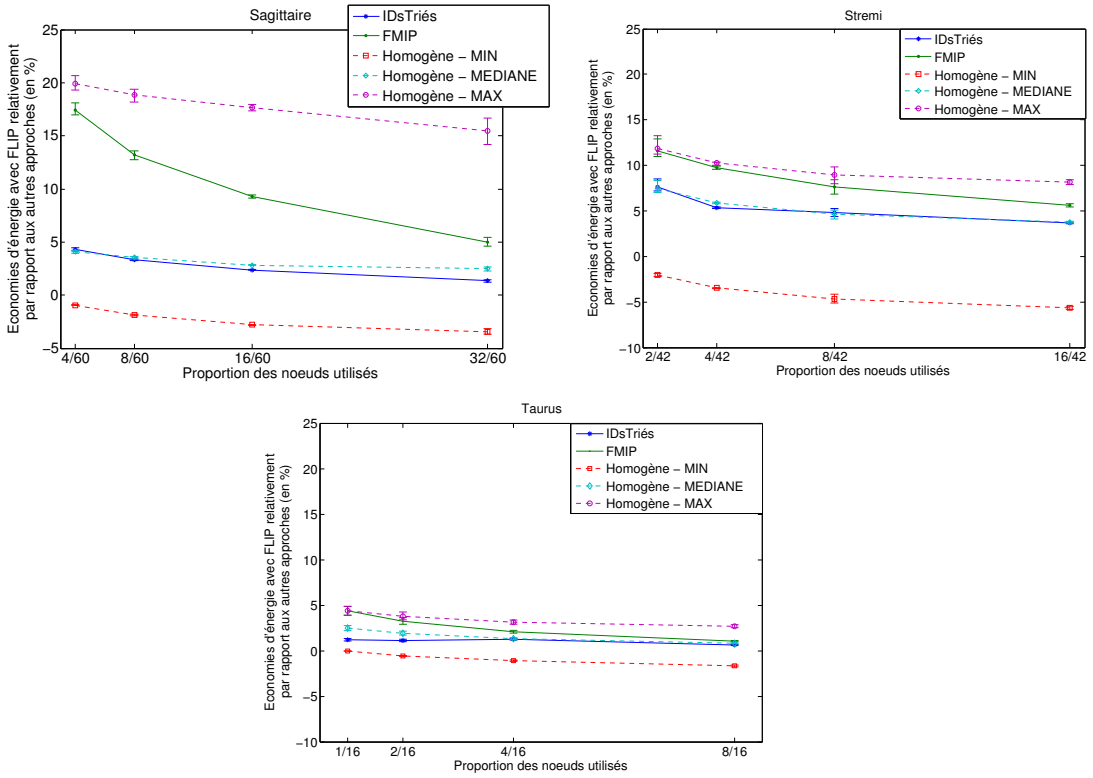

FIGURE 5.4 – Proportion de l'énergie économisée avec *FLIP* par rapport aux autres politiques d'attribution de noeuds sur les trois grappes de calcul

En outre, nous observons que les courbes représentant *Homogène_ MEDIANE* et *IDsTries* sont très proches et sont quasiment superposées. Cela montre que ces deux politiques d'attribution de noeuds sont comparables en termes de consommation énergétique. En effet, même si la politique *Homogène_ MEDIANE* n'est pas réaliste, la politique fournit une approximation des mesures énergétiques plus raisonnable que *Homogène_ MIN* et *Homogène_ MAX*. Les politiques *Homogène* ne sont pas appropriées spécialement lorsque l'hétérogénéité électrique est aussi importante que celle que nous pouvons observer pour les grappes *Sagittaire* et *Stremi* mais aussi lorsque nous mesurons un noeud non représentatif de la plate-forme comme c'est le cas pour *Homogène_ MIN* et *Homogène_ MAX*. Même si *Homogène_ MEDIANE* est la politique *Homogène* qui s'apparente le plus à la réalité, elle économise moins d'énergie que*FLIP* car elle n'utilise pas en priorité les noeuds les moins consommateurs. Par conséquent, afin de mesurer la consommation énergétique d'une application exécutée sur un ensemble de noeuds, il est vraiment nécessaire de mesurer la consommation énergétique de chaque noeud.

De plus, nous observons que pour les trois grappes, *Homogène_ MIN* est le seul scénario où nous observons des économies d'énergies négatives. En effet, c'est le scénario irréel idéal où tous les noeuds consomment autant que le noeud qui consomme le moins d'énergie. Nous observons également que la proportion d'énergie économisée est toujours la plus importante lorsque nous comparons l'approche *FLIP* à *Homogène_ MAX*. En fait, *Homogène_ MAX* correspond au pire scénario irréel étant donné que nous mesurons arbitrairement la puissance électrique du noeud le plus consommateur et nous considérons que tous les autres noeuds ont la même puissance électrique.

5.2.3 Évaluation du gain en puissance électrique lors de l'extinction des noeuds inutilisés

Dans cette section, nous évaluons les gains en termes de puissance électrique lorsque le propriétaire de la plate-forme décide d'éteindre les noeuds inutilisés. Pour cela, nous calculons les économies de puissance électrique que nous obtenons en éteignant les noeuds inutilisés avec l'approche *FLIP* en comparaison aux économies obtenues avec les autres approches d'attribution de noeuds. Comme annoncé dans la section 5.2.1, ceci représente le point de vue du propriétaire de l'infrastructure et est valable dans le cas où ce dernier facture l'utilisateur en fonction du nombre de noeuds.

La figure 5.5 montre la puissance électrique économisée sur chaque grappe si on éteint les noeuds inutilisés avec l'approche *FLIP* au lieu d'éteindre les noeuds inutilisés avec les autres approches d'attribution de noeuds. L'axe des abscisses représente le nombre de noeuds inutilisés qui ont été éteints. L'axe des ordonnées représente la différence de puissance électrique économisée si on éteint les noeuds inutilisés avec *FLIP* au lieu d'éteindre les noeuds avec une autre politique d'attribution de noeuds. Chaque graphique correspond à l'une des trois grappes considérées. Les axes des ordonnées des trois graphiques ont une même échelle.

Tout d'abord, nous observons qu'en fonction de la grappe, la puissance électrique économisée grâce à l'approche *FLIP* est plus ou moins importante. Dans la grappe *Taurus*, la puissance électrique économisée semble être négligeable (moins de 20 W) en comparaison aux grappes *Sagittaire* et *Stremi* (plus de 190 W). La raison est que la grappe *Taurus* est beaucoup moins hétérogène électriquement en comparaison aux grappes *Sagittaire* et *Stremi*. C'est aussi celle qui contient le moins de noeuds de calcul.

Les courbes pour les politiques réalistes d'attribution de noeuds restent toujours positives. Cela signifie que la politique d'attribution de noeuds *FLIP* permet de consommer moins de puissance électrique que les autres politiques réalistes. Par exemple, comparée à l'approche *FMIP*, l'approche *FLIP* permet d'économiser jusqu'à 405 W sur la grappe *Sagittaire*, jusqu'à 195 W sur la grappe *Stremi* et moins de 20 W sur la grappe *Taurus*. En effet, ceci est dû au fait que les noeuds restants inutilisés font partie des noeuds les plus consommateurs. Ainsi, l'approche *FLIP* permet au propriétaire de la plate-forme de consommer moins lorsqu'il éteint les noeuds

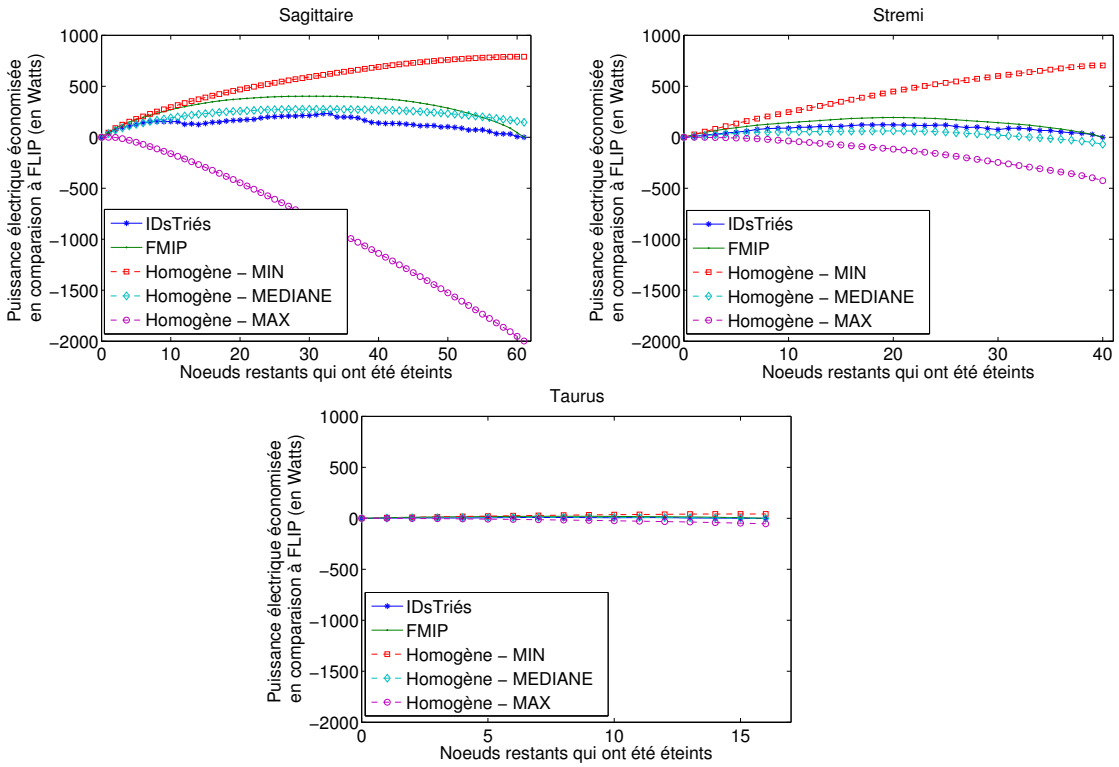

FIGURE 5.5 – Puissance électrique économisée (en Watts) lorsque nous éteignons les noeuds inutilisés avec *FLIP* en comparaison aux autres politiques d'attribution de noeuds sur les trois grappes de calcul considérées

inutilisés.

Par ailleurs, nous observons qu'en ce qui concerne les politiques réalistes d'attribution de noeuds, nous n'économisons pas d'énergie lorsqu'aucun noeud n'est éteint ($x = 0$) ou lorsque tous les noeuds sont éteints. Comparées à *FMIP*, les économies d'électricité grâce à *FLIP* augmentent quand le nombre de noeuds utilisés croît et que la grappe est utilisée à moins de 50 %. Ensuite, à partir de 50% d'utilisation de la grappe, l'électricité économisée décroît quand le nombre de noeuds utilisés augmente. En fait, jusqu'à la moitié de la taille de la grappe, il n'y a pas de recouvrement (*i.e.,* pas les même noeuds) entre la liste ordonnée obtenue avec *FLIP* et celle obtenue avec *FMIP*. Cela revient donc à comparer l'extinction des noeuds les plus extrêmes (les moins consommateurs *vs* les plus consommateurs). À partir de 50 %, il commence à y avoir de plus en plus de recouvrement entre les listes ordonnées obtenues avec les différentes politiques réalistes d'attribution de noeuds. Cela revient à éteindre en partie les mêmes noeuds.

En ce qui concerne *IDsTries*, nous observons une autre réalité. Avec un nombre de noeuds éteints qui augmente, la puissance électrique économisée tantôt augmente tantôt diminue. Ceci est dû au fait que la liste des noeuds obtenue avec *IDsTries* est désordonnée et peut donc chevaucher avec la liste obtenue avec *FLIP* quel que soit le nombre de noeuds éteints. Par exemple, le premier noeud éteint avec *IDsTries* peut être le même que celui éteint avec *FLIP* si le premier noeud de la plate-forme se trouve être celui qui consomme le plus.

Homogène_MAX est le seul scénario où nous observons des économies négatives : il économise de l'électricité de façon illusoire car les noeuds éteints sont tous considérés comme consommant autant d'énergie que le noeud qui consomme le plus dans la grappe de calcul. Nous constatons également que plus le nombre de noeuds éteints augmente, plus l'électricité économisée avec *FLIP* devient conséquente en comparaison à *Homogène_MIN*. Ces scénarios irréalistes permettent de borner relativement à *FLIP*, le gain et la perte maximum d'énergie en considérant un noeud particulier de la grappe.

5.3 Conclusions du chapitre

Ce chapitre a présenté différentes solutions pour réduire la consommation énergétique lors des applications de calcul très haute performance. En effet, la première solution pour consommer moins d'énergie consiste à se baser sur les estimations des consommations énergétiques des différentes versions (protocoles ou algorithmes) des services applicatifs. **En fournissant ces estimations énergétiques avant l'exécution de l'application**, nous avons montré qu'il était possible de **choisir le protocole de tolérance aux pannes ou l'algorithme de diffusion de données qui consomme le moins d'énergie, et ce avant l'exécution de l'application, en fonction du contexte d'exécution et de l'application envisagés**. La mise en place de cette solution nécessite de fournir à l'utilisateur avant l'exécution de son application, les estimations des consommations énergétiques des différentes versions

du service applicatif qu'il utilise.

La consommation énergétique est un des critères pour le choix d'une version (protocole ou algorithme) d'un service applicatif mais ce n'est bien évidemment pas le seul. En fonction de ses besoins, l'utilisateur peut établir un choix bi-critères en s'appuyant par exemple sur la consommation énergétique et sur la performance (*i.e.,* temps d'exécution). En effet, comme nous avons pu le voir notamment dans le cas des algorithmes de diffusion de données, la consommation énergétique d'un service applicatif n'est pas toujours liée à son temps d'exécution. L'estimateur de la consommation énergétique que nous proposons est également capable d'estimer les temps d'exécution des différentes opérations que nous retrouvons dans les protocoles de tolérance aux pannes et dans les algorithmes de diffusion de données. En fournissant également les temps d'exécutions des différentes opérations liées à la tolérance aux pannes ou à la diffusion de données, l'estimateur de la consommation énergétique que nous proposons permettra ainsi à l'utilisateur de faire un choix en fonction de ces deux critères.

L'autre solution pour consommer moins d'énergie consiste à exploiter les différences de puissance électrique qui caractérisent les noeuds identiques d'une même grappe et ce en privilégiant l'utilisation des noeuds de calcul dont la puissance électrique au repos est la moindre. Cette approche baptisée *FLIP* s'oppose aux approches irréalistes qui consistent à considérer à tort que des noeuds appartenant à des grappes de calcul homogènes consomment la même puissance électrique lorsqu'ils sont au repos ou lorsqu'ils exécutent les mêmes programmes. Avec de telles approches irréalistes, les utilisateurs choisiraient des noeuds quel qu'ils soient en pensant que de toutes façons la consommation totale c'est la consommation d'un noeud multipliée par le nombre de noeuds. La mise en place de cette solution nécessite une mesure des puissances électriques au repos des différents noeuds de la plate-forme de calcul très haute performance afin de les trier dans l'ordre croissant des puissances électriques.

D'une part, nous avons montré qu'en choisissant d'exécuter une application sur les noeuds qui consomment le moins, l'utilisateur peut réduire significativement la consommation énergétique (jusqu'à 17 % sur *Sagittaire*) de ces applications et donc sa facture. Moins il utilise de noeuds, plus les économies d'énergie grâce à *FLIP* seront notables en comparaison aux économies obtenues avec les autres approches. Si son application utilise toute la plate-forme, il ne réduira pas sa consommation énergétique dans la mesure où il utilisera tous les noeuds quel que soit la politique d'attribution de noeuds. Ainsi, s'il est facturé en fonction de sa consommation énergétique, *FLIP* lui permettra d'économiser de l'énergie et donc de l'argent. Avec une facturation sur la consommation énergétique, *FLIP* sera plus profitable pour les premiers utilisateurs car eux ils peuvent faire un choix sur un plus large éventail de noeuds. Les derniers utilisateurs auront un choix plus restreint et seront donc perdants étant donné qu'ils n'auront le choix que parmi les noeuds qui consomment les plus. Cette politique de facturation est donc analogue à celle qui est généralement utilisée pour la tarification des chambres d'hôtels et des billets de transport (avion,

train, etc.) dans la mesure où ceux qui réservent leurs places en premier ont le plus de chance d'avoir les tarifs les plus bas. Cette politique de facturation est fondée sur les théories du *"yield management"*. À l'opposé, s'il est facturé en fonction du nombre de noeuds, il n'y aura aucune incidence sur sa facture s'il choisit d'utiliser des noeuds plutôt que d'autres.

D'autre part, nous avons montré qu'en attribuant à l'utilisateur les noeuds qui consomment le moins, le propriétaire de la plate-forme peut économiser plus d'électricité s'il décide d'éteindre les noeuds restants inutilisés (c'est-à-dire les noeuds qui consomment le plus). Dans des travaux précédents comme [Orgerie and Lefèvre, 2011], les auteurs ont proposé d'éteindre les noeuds qui ne sont pas utilisés. Cependant étant donné que ces travaux considèrent que tous les noeuds d'une même grappe de calcul consomment la même puissance électrique, aucun d'eux ne met en évidence que le choix des noeuds à éteindre peut faire évoluer les économies d'énergie. Pour que *FLIP* soit financièrement profitable au propriétaire de l'infrastructure, il faut que celui-ci facture l'utilisateur en fonction du nombre de noeuds.

En somme, le choix des noeuds à attribuer lors des demandes de réservations de ressources de calcul est un des critères qu'il faut prendre en compte dans les algorithmes d'ordonnancement. Pour réduire la consommation énergétique dans les infrastructures de calcul haute performance, l'approche *FLIP* suggère de choisir parmi les noeuds disponibles ceux dont la puissance électrique au repos est la moindre. En fonction de la politique de facturation, l'approche *FLIP* est une approche d'attribution de noeuds financièrement profitable, soit pour l'utilisateur soit pour le propriétaire de la plate-forme.

Au delà de ces deux solutions, grâce aux mesures de la puissance électrique, la compréhension du comportement électrique des différentes versions d'un service applicatif permet d'envisager d'autres solutions pour réduire la consommation énergétique d'un protocole ou d'un algorithme donnés. En effet, en prédisant les périodes d'inactivé et les périodes d'attente active, il est possible d'envisager l'application de leviers énergétiques tels le ralentissement de ressources (DVFS) voire l'extinction de quelques composants si les périodes d'inactivité ou d'attentes actives sont suffisamment longues.

6 Consommer mieux dans les infrastructures de calcul très haute performance

Les supercalculateurs sont devenus de gros consommateurs d'électricité. Selon la liste du TOP 500 publiée en juin 2013, les infrastructures de calcul intensif les mieux classées en termes de performance consomment déjà quelques mégawatts. Au delà de la consommation énergétique très élevée de ces systèmes, les coûts financiers et les problèmes écologiques liés à ces infrastructures sont un véritable frein à leur déploiement.

Les solutions que nous avons présentées dans le chapitre précédent visent à "consommer moins" d'énergie, en réduisant la consommation énergétique des infrastructures de calcul très haute performance et des services applicatifs que ces infrastructures exécutent. Nous avons également besoin de solutions pour "consommer mieux" cette énergie [Diouri et al., 2011] en la consommant quand elle est disponible à moindre coût financier et à moindres émissions de CO_2.

À cette fin, et dans la perspective de faciliter la coordination de toutes ces solutions, nous recourons à la conception d'un environnement logiciel unifié appelé SESAMES : *Smart and Energy-aware Service oriented Architecture Manager at Extreme-Scale* [Diouri et al., 2012a, Diouri et al., 2013c, Diouri et al., 2013b]. Cet environnement logiciel permet d'agréger les solutions pour "consommer moins" d'une part et les solutions pour "consommer mieux" d'autre part. Il s'appuie donc sur les mesures et estimations que nous avons présentées dans les chapitres précédents. Jusque là, nous avons eu besoin d'interactions principalement avec l'utilisateur et l'administrateur de la plate-forme de calcul très haute performance. Dans ce chapitre, nous introduisons une interaction avec un nouvel acteur : le fournisseur

d'énergie afin de mettre en adéquation l'usage de la plate-forme à la fourniture de l'électricité qui a un caractère dynamique.

Dans la section 6.1, nous présentons l'architecture et les interactions de SESAMES avec les différents acteurs (utilisateur, plate-forme, administrateur et fournisseur d'énergie). La section 6.2 généralise l'attribution des noeuds aux utilisateurs (approche *FLIP*) que nous avons introduit dans le chapitre précédent en proposant un ordonnancement vert multi-critères des réservations de ressources qui prend en compte également les critères de coût financier et de l'impact environnemental. La section 6.3 présente des résultats obtenus par simulation qui montrent comment notre algorithme d'ordonnancement multi-critères permet de réduire la consommation énergétique tout en diminuant l'impact environnemental et le coût financier, et tout en respectant les contraintes de l'utilisateur et du fournisseur d'énergie.

6.1 SESAMES : un environnement logiciel pour consommer moins et mieux

Le but de SESAMES est de gérer l'exécution des applications de calcul très haute performance sur les actuels et futurs supercalculateurs dans une perspective de mieux consommer l'énergie. Dans cette section, nous revenons en partie sur quelques composants que nous avons pu introduire dans les chapitres précédents et présentons les nouveaux composants qui visent à consommer mieux.

6.1.1 Architecture conceptuelle de SESAMES

Puisque les infrastructures de calcul très haute performance sont de gros consommateurs d'énergie et que leur consommation est irrégulière dans le temps [Orgerie et al., 2008a], SESAMES propose d'établir une communication continue avec le fournisseur d'énergie. De tels flux de communication visent à optimiser la production, la distribution mais aussi la consommation d'énergie. La figure 6.1 présente l'architecture globale de l'infrastructure considérée. D'une part, les fournisseurs d'énergie adaptent l'approvisionnement en énergie en fonction des besoins des gros consommateurs (tels que les supercalculateurs). D'autre part, les utilisateurs des infrastructures distribuées à large échelle préféreront consommer l'énergie lorsqu'elle est la moins coûteuse et/ou la plus "verte". Par "verte", nous insinuons que la production de l'énergie est moins polluante et ainsi génère une quantité réduite en émissions de CO_2 (comme c'est le cas par exemple pour les énergies solaire et éolienne).

Par ailleurs, dans l'optique de réduire la consommation énergétique globale, SESAMES agit directement sur les noeuds du supercalculateur. Un wattmètre est branché sur chaque noeud et mesure sa puissance électrique. Nous pouvons imaginer que les unités de distribution électrique (PDUs) fourniront des mesures aussi précises et avec une fréquence aussi élevée que celles que nous relevons à l'aide de nos wattmètres (tels que OmegaWatt et WattsUp). Ceci permettrait de mesurer plus facilement la puissance électrique de chaque noeud.

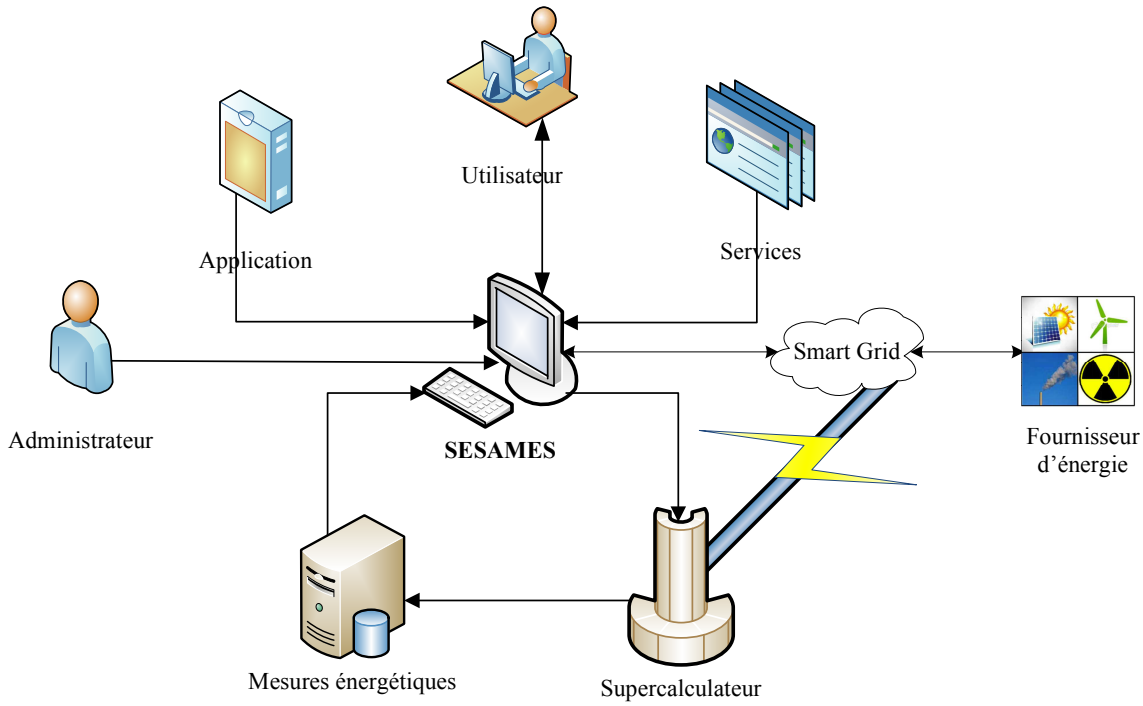

FIGURE 6.1 – Interactions de SESAMES avec les différents acteurs

Dans la section 6.1.2, nous spécifions ce que nous voulons dire par service et nous détaillons les différents services que l'on peut considérer dans SESAMES.

6.1.2 Services pris en compte dans SESAMES

La figure 6.2 présente des exemples de services qui nous semblent incontournables dans le domaine du calcul très haute performance et sur lesquels SESAMES peut aider l'utilisateur à choisir la meilleure version du service en intégrant les contraintes énergétiques.

Parmi les services considérés dans SESAMES, nous avons étudié dans le chapitre 4.3 quelques services relatifs à la tolérance aux pannes et aux échanges collectifs de données. En ce qui concerne la tolérance aux pannes, SESAMES inclut donc deux services. D'une part, l'un est dédié à la phase de sauvegarde de points de reprise (*i.e., checkpointing*) qui est réalisée pendant le fonctionnement sans panne d'une application [Diouri et al., 2013e]. Cela consiste à sauvegarder une image de l'actuel état de l'application [Chandy and Lamport, 1985]. D'autre part, l'autre est dédié à la phase de récupération de points de reprise (*i.e., recovery*) dans le cas d'une défaillance. Cela consiste à récupérer le dernier point de reprise sauvegardé.

En ce qui concerne les opérations collectives d'échanges de données, SESAMES incorpore les services suivants :

– *Scatter*, qui consiste à distribuer à plusieurs processus des portions de données issues d'un processus source ;

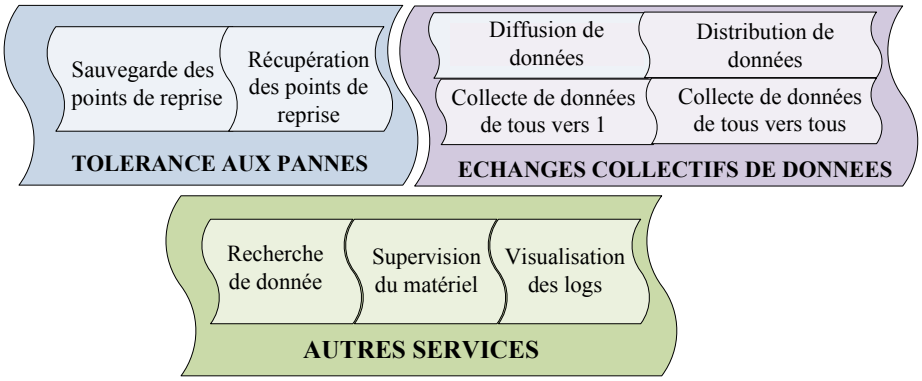

FIGURE 6.2 – Services pris en compte dans SESAMES

- *Gather*, qui consiste à récupérer sur un processus les données issues de plusieurs processus ;
- *Broadcast*, la diffusion de données à tous les processus [Diouri et al., 2013g] ;
- *All Gather*, qui consiste à récupérer les données de tous les processus sur tous les processus.

D'autres services sont également à prendre en compte : par exemple, un service pour rechercher une donnée parmi tous les processus et un service pour visualiser en temps réel les logs d'une application. Superviser les ressources matérielles qui sont mises en jeu dans un système distribué à très large échelle est un autre service important qui est requis si par exemple nous souhaitons visualiser en temps réel la consommation énergétique de chacun des noeuds.

Chacun de ces services peut-être implémenté à l'aide de différentes versions de protocoles ou d'algorithmes. Par exemple, en ce qui concerne la diffusion de données, nous avons décliné dans le chapitre 4, quatre algorithmes différents de diffusion de données. Un des objectifs de SESAMES est d'aider les utilisateurs à choisir la version la plus convenable (*i.e.,* protocole ou algorithme) du service. Nous avons montré dans le chapitre 4 comment à partir d'un calibrateur et d'un estimateur de la consommation énergétique d'un service, nous pouvions guider l'utilisateur et l'administrateur dans leurs choix. La section suivante présente les composants de SESAMES ainsi que leurs interactions avec les différents acteurs.

6.1.3 Composants de SESAMES

Concevoir un environnement logiciel unifié efficace en énergie est indispensable pour favoriser un dialogue et une coordination entre les différents acteurs intervenant sur l'infrastructure de calcul très haute performance mais également entre les différents composants utilisés dans une perspective d'améliorer la consommation énergétique. La figure 6.3 présente les principaux composants de cet environnement logiciel.

De plus, SESAMES inclut un ordonnanceur vert et multi-critères des réserva-

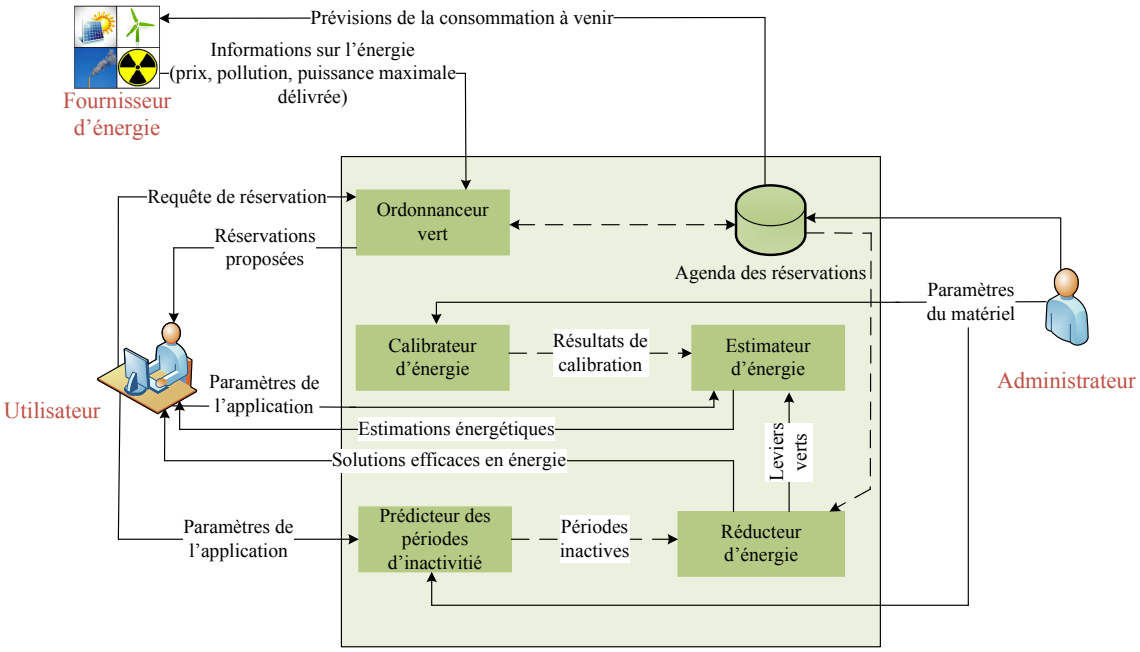

FIGURE 6.3 – Les différents composants de SESAMES

tions de ressources de calcul faites par les utilisateurs. Cet ordonnanceur est en charge de planifier efficacement les nouvelles demandes de réservation et d'allouer les ressources nécessaires pour chaque réservation. Pour cela, l'ordonnanceur vert tente de résoudre un problème d'optimisation multi-critères en tenant compte de plusieurs contraintes. Comment allouer les noeuds du supercalculateur au moment le plus proche de celui souhaité par l'utilisateur, en consommant le moins d'énergie possible, de l'énergie moins polluante, au plus bas coût financier et sans dépasser la limitation de puissance fixée par le fournisseur d'énergie ? Une fois que la réservation est placée, l'ordonnanceur vert met à jour l'agenda des réservations de SESAMES.

Étant donné que les supercalculateurs à très large échelle ont une consommation énergétique importante et irrégulière, SESAMES fournit la possibilité d'estimer avec précision la consommation énergétique des différents services présentés dans la section 6.1.2 qui sont incontournables dans les applications qui s'exécuteront à l'échelle exaflopique. Afin de prendre en compte les différentes spécificités techniques du matériel, l'estimateur de l'énergie s'appuie sur la calibration de plate-forme d'exécution pour les différents services (voir chapitre 4.3) Actuellement, l'estimation d'énergie avec SESAMES est proposée pour la tolérance aux pannes [Diouri et al., 2013e] et pour la diffusion des données [Diouri et al., 2013g] (voir chapitre 4.3).

Pour optimiser la consommation énergétique des supercalculateurs, SESAMES intègre un réducteur de la consommation énergétique qui est en charge de diminuer la consommation énergétique des services et des applications en proposant d'appliquer des leviers énergétiques au niveau des composants matériels : éteindre ou

ralentir des composants (noeud de calcul, processeur, mémoire RAM, disque dur, etc.) lorsqu'ils sont au repos ou en attente active. Le réducteur d'énergie propose ces leviers verts en fonction des périodes d'inactivité et d'attente active connues par les noeuds de calcul et des droits pour éteindre, allumer ou ralentir les ressources de calcul, qui sont attribués par l'administrateur de l'infrastructure à l'utilisateur. Par exemple, avec la récupération non coordonnée des points de reprise (section 6.2), seuls les processus défaillants redémarrent et lancent la restauration des points de reprise pendant que les processus non-défaillants restent en attente active [Bouteiller et al., 2006]. Dans une perspective de réduire la consommation énergétique de la récupération non coordonnée, nous proposons soit d'éteindre soit de ralentir les coeurs de calcul des processus non défaillants pendant toute la phase de récupération des processus défaillants. La durée de cette attente active est très variable : elle peut aller de quelques secondes à plusieurs minutes. Il est donc nécessaire de prédire la durée de cette attente active notamment pour savoir si elle est suffisamment longue (supérieure au T_s [Orgerie et al., 2008b]) pour éteindre et rallumer les noeuds de calcul. Dans le cas contraire, on pourra toujours se contenter de ralentir les noeuds de calcul en appliquant les techniques DVFS. Ces solutions vertes sont elles-mêmes ré-évaluées par l'estimateur énergétique afin d'informer l'utilisateur de l'importance de l'énergie que l'on peut économiser grâce à ces solutions.

6.1.4 Grille électrique intelligente

Les supercalculateurs à très large échelle consomment énormément d'électricité et sont considérés par les fournisseurs d'énergie comme d'importants clients dans la mesure où ils consomment quelques mégawatts. En effet, selon le classement du TOP 500 publié en juin 2013, plus de 40 supercalculateurs consomment plus d'un mégawatt. Afin d'établir une meilleure communication entre les fournisseurs d'énergie et ces importants clients, SESAMES s'appuie sur une grille électrique intelligente ("smart grid", voir section 2.2.2.1 du chapitre 2).

6.1.4.1 Du fournisseur d'énergie vers SESAMES

Grâce à des flux de communication permanents, SESAMES recueille les informations suivantes auprès du fournisseur d'énergie :

- l'agenda du coût financier de l'énergie (kWh) afin de connaître/calculer le prix de l'électricité consommée ;
- l'agenda des émissions de CO_2 correspondant aux types d'énergies utilisées par le fournisseur d'énergie pour alimenter le supercalculateur, afin de connaître l'impact environnemental de l'énergie consommée ;
- l'agenda des limitations d'électricité fournie (quantité maximale disponible) afin de connaître la puissance électrique maximale que le fournisseur d'énergie peut délivrer [Fan et al., 2007b].

En effet, les fournisseurs d'énergie proposent souvent des prix par kWh en fonction de l'heure et du jour où cette énergie est consommée. Par exemple, EDF [1], le

1. EDF : http://bleuciel.edf.com

fournisseur d'énergie en France affiche des prix en euros par kWh d'une part du jour de consommation (jour "bleu", "blanc" ou "rouge") et d'autre part des horaires dans la journée considérée (période creuse, de 22h à 6h ou période pleine, de 6h à 22h). Le prix par kWh dans les périodes pleines des jours "rouges" est plus de sept fois celui des périodes creuses dans les jours "bleus". C'est loin d'être négligeable !

En outre, l'énergie fournie peut provenir de multiples sources : pétrole, gaz naturel, charbon, soleil, vent, etc. Ainsi, l'impact environnemental (émissions de CO_2) est fonction de la ressource à partir de laquelle l'énergie a été produite. La puissance électrique maximale que peut délivrer le fournisseur d'énergie peut varier dans le temps. Ainsi, SESAMES peut adapter l'ordonnancement des réservations de ressources en fonction de cette limitation d'électricité fournie.

Grâce aux informations relatives au coût financier et à l'impact environnemental de l'énergie consommée récupérées auprès du fournisseur d'énergie, SESAMES peut également se baser sur les estimations de consommation énergétique pour évaluer le coût financier et la quantité de CO_2 émise pour chacune des versions du service applicatif que l'utilisateur souhaite utiliser. Ainsi, l'utilisateur pourrait se voir proposer par SESAMES plusieurs créneaux horaires pour exécuter son application tout en cherchant à réduire les coûts financiers et environnementaux liés à son service applicatif.

6.1.4.2 De SESAMES vers le fournisseur d'énergie

Grâce aux informations issues du calibrateur et de l'estimateur, SESAMES peut prévoir la consommation future du supercalculateur en fonction des réservations planifiées et informer en conséquence le fournisseur d'énergie de cette consommation à venir. Dans le cas où il serait prévu que le supercalculateur consomme moins d'énergie que ce dont dispose le fournisseur d'énergie, SESAMES avertit ce dernier de sorte qu'il puisse réduire ou arrêter la production des énergies issues de sources polluantes (charbon, pétrole, etc.). Dans le cas inverse, si le supercalculateur prévoit des pics élevés en termes de puissance électrique, SESAMES demande au fournisseur d'énergie de délivrer plus de puissance.

6.2 Ordonnancement efficace en énergie des réservations des ressources

Grâce aux informations provenant du fournisseur d'énergie et à notre connaissance énergétique de la plate-forme, SESAMES planifie les réservations des noeuds dans l'optique de consommer mieux. L'ordonnanceur vert des réservations de SESAMES maintient un agenda des réservations pour chaque noeud. L'agenda enregistre toutes les réservations relatives à un noeud.

Afin d'exécuter leurs applications, les utilisateurs envoient à SESAMES une requête de réservation de plusieurs noeuds du supercalculateur. Une demande de réservation consiste en un nombre N de noeuds requis, une durée de réservation D,

une date de début au plus tôt $t_{d_{min}}$ et une date de début au plus tard $t_{d_{max}}$. Afin de planifier une réservation, l'ordonnanceur vert de SESAMES résout le problème (P) d'optimisation multi-critères suivant :

Déterminer la date de début (placement) de la réservation t_0 telle que :
- C_1 : t_0 est compris entre $t_{d_{min}}$ et $t_{d_{max}}$;
- C_2 : au moins, N noeuds sont disponibles entre t_0 et $t_0 + D$;
- C_3 : la puissance électrique maximale $P_{capping}$ dont dispose le fournisseur d'énergie n'est pas dépassée ;
- O_1 : l'énergie $\xi(t_0, t_0 + D)$ consommée pendant la réservation est minimisée ;
- O_2 : le coût financier $\zeta(t_0, t_0 + D)$ de la réservation est minimisée ;
- O_3 : l'impact environnemental (quantité de CO_2 émise) $\rho(t_0, t_0 + D)$ de la réservation est minimisée.

C_1, C_2 et C_3 sont les contraintes du problème P.

O_1, O_2 et O_3 sont les objectifs du problème P.

Comme spécifié dans la section 6.1.4.1, l'impact environnemental $\rho(t)$ dépend de la source d'énergie qu'approvisionne le fournisseur d'énergie. Il est évalué en considérant la quantité de CO_2 émise par joule consommé. De façon analogue, le coût financier $\zeta(t)$ dépend de l'agenda du coût financier de l'énergie délivrée par le fournisseur d'énergie. Il est évalué en considérant le prix du joule.

Nous notons $P^{N_{res}}_{MAX}(t, t + D)$ la puissance électrique maximale consommée par les N_{res} noeuds qui sont réservés entre t et $t + D$. Pour évaluer $P^{N_{res}}_{MAX}(t, t + D)$, SESAMES mesure la puissance électrique maximale P^i_{max} pour chaque noeud i. P^i_{max} est obtenue en mesurant la puissance électrique du noeud i lorsqu'il exécute `cpuburn` (voir chapitre 3) sur chacun des coeurs de calcul du noeud. $P^{N_{res}}_{MAX}(t, t+D)$ est égale à $max_{entretett+D}(\sum_{i=1}^{N_{res}} P^i_{max})$ où P^i_{max} est la puissance électrique maximale du noeud i. Ainsi, la puissance électrique de chaque noeud qui exécute une application donnée est comprise entre P^i_{repos} et P^i_{max}. P^i_{repos} est obtenue en mesurant la puissance électrique de chaque noeud i pendant qu'il n'exécute que le système d'exploitation.

Nous considérons que pour un noeud i, la puissance électrique moyenne pendant D est $\alpha_i P^i_{max}$, où α_i est une coefficient compris entre $\frac{P^i_{repos}}{P^i_{max}}$ et 1. Les mesures de P^i_{repos} et de P^i_{max} ont besoin d'être relevées qu'une seule fois et font partie du processus de la calibration de la plate-forme d'exécution.

Pour le calcul des différents fonctions objectifs $\xi(t, t+D)$, $\zeta(t, t+D)$ et $\rho(t, t+D)$, nous ne prenons en compte que les N noeuds disponibles qui consomment le moins entre t et $t + D$. $\xi(t, t + D)$, $\zeta(t, t + D)$ et $\rho(t, t + D)$ sont tels que :

$$\xi(t, t+D) = \sum_{i=1}^{N}(\alpha_i P^i_{max} D) = D \cdot \frac{\sum_{i=1}^{N}(P^i_{max})}{N} \cdot \sum_{i=1}^{N}(\alpha_i)$$

$$\zeta(t, t+D) = \xi(t, t+D) \times \frac{\int_t^{t+D} \zeta(t)\,\mathrm{d}t}{D}$$

$$\rho(t, t+D) = \xi(t, t+D) \times \frac{\int_t^{t+D} \rho(t)\,\mathrm{d}t}{D}$$

Nous obtenons la première équation en appliquant la propriété fondamentale du barycentre sur le système $\{(P^i_{max}, \alpha_i), i \in [1, n]\}$. $\sum_{i=1}^{n}(\alpha_i)$ dépend uniquement de l'application qui sera exécutée pendant la réservation.

Nous résolvons le problème (P) à l'aide de l'algorithme 1.

Le dialogue entre l'utilisateur et l'ordonnanceur vert de SESAMES est présenté dans la figure 6.4. Une fois que la demande de réservation est exprimée par l'utilisateur, SESAMES informe l'utilisateur des différentes possibilités :

- la réservation demandée n'est pas possible ; dans ce cas, l'utilisateur peut essayer de demander une nouvelle réservation en changeant ses critères ;
- il y a une date de début de la réservation t_0 qui minimise simultanément $\xi(t_0, t_0 + D)$, $\zeta(t_0, t_0 + D)$ et $\rho(t_0, t_0 + D)$; dans ce cas, l'utilisateur confirme ou non la réservation optimale proposée par SESAMES ;
- il y a six dates de début de la réservation distinctes ($t_{\xi\zeta\rho}$, $t_{\xi\rho\zeta}$, $t_{\rho\xi\zeta}$, $t_{\rho\zeta\xi}$, $t_{\zeta\rho\xi}$ ou $t_{\zeta\xi\rho}$) qui minimisent ξ, ζ et ρ en donnant des poids différents à ces trois critères. Ainsi, en considérant les différentes valeurs de ξ, ζ et ρ, l'utilisateur peut choisir l'une de ces six dates. En d'autres termes, cela signifie que SESAMES demande à l'utilisateur de choisir un classement des critères de la consommation énergétique, du coût financier, et l'impact environnemental.

6.3 Evaluation de l'ordonnanceur de réservations multi-critères

Cette section présente l'environnement de simulation et les résultats de l'évaluation de l'ordonnanceur vert de réservations multi-critères. L'objectif est de comparer notre approche d'ordonnancement (avec un classement donné pour les trois critères) à l'approche d'ordonnancement qui consiste à démarrer "dès que possible" la réservation de l'utilisateur.

6.3.1 Environnement de simulation

Pour évaluer les avantages de l'ordonnanceur vert de réservations multi-critères, nous l'avons simulé sous Matlab afin de le comparer à l'approche qui consiste à démarrer "dès que possible" la réservation de l'utilisateur.

Algorithm 1 Résolution du problème d'optimisation multi-objectifs (P)

$t_{tmp}[]$: tableau d'entiers ; nb, t : entiers ;
$nb \leftarrow 0$;
*/*** Détermination des dates de début possibles t, i.e., celles qui satisfont les contraintes C_1, C_2 et C_3 ***/*
/ On parcourt un intervalle d'entiers */*
for $t \in [t_{d_{min}}..t_{d_{max}}]$ **do**
  if [(au moins N noeuds sont disponibles pendant $[t, t+D]$) et (les N noeuds qui consomment le moins d'énergie sont tels que : $P_{MAX}^{N_{res}+N}(t, t+D) < P_{capping}$)] **then**
    nb_{++} ;
    $t_{tmp}[nb] \leftarrow t$;
  end if
end for

if (nb == 0) **then**
  Informer l'utilisateur que la réservation n'est pas possible, car il n'y a pas assez de noeuds ou il y aura dépassement de la limitation de puissance électrique.
  return -1
else
  $\xi[], \zeta[], \rho[]$: tableaux de décimaux ;
  */*** Calcul des valeurs des différentes fonctions objectifs pour toutes les dates de début possibles t ***/*
  for $i \in [1..nb]$ **do**
    $\xi[i] \leftarrow \xi(t_{tmp}[i], t_{tmp}[i] + D)$;
    $\zeta[i] \leftarrow \zeta(t_{tmp}[i], t_{tmp}[i] + D)$;
    $\rho[i] \leftarrow \rho(t_{tmp}[i], t_{tmp}[i] + D)$;
  end for
end if

x, y, z : entiers ;
$Min\xi, Min\zeta, Min\rho$: *decimaux* ; */* Valeurs minimales pour chaque fonction */*
$TMin\xi[], TMin\zeta[], TMin\rho[]$: tableaux d'entiers ;
/ Obtenir tous les t_{tmp} minimisant chaque fonction objectif */*
/ Définition de Obtenir_ T_ Min() dans l'algorithme 2 */*
$(Min\xi, TMin\xi, x) \leftarrow Obtenir\_T\_Min(\xi, t_{tmp}, nb)$;
$(Min\zeta, TMin\zeta, y) \leftarrow Obtenir\_T\_Min(\zeta, t_{tmp}, nb)$;
$(Min\rho, TMin\rho, z) \leftarrow Obtenir\_T\_Min(\rho, t_{tmp}, nb)$;

/ S'il existe, retourner la valeur du plus petit $t_{tmp}[i]$ qui minimise O_1, O_2 et O_3 */*
for $i \in [1..nb]$ **do**
  if [$(t_{tmp}[i] \in TMin\xi)$ et $(t_{tmp}[i]) \in TMin\zeta)$ et $(t_{tmp}[i]) \in TMin\rho)$] **then return** $t_{tmp}[i]$, $Min\xi$, $Min\zeta$ et $Min\rho$;
  end if
end for

/* *S'il n'existe pas : fournir toutes les possibilités en fonction des 6 classements possibles pour les 3 critères* */
$t_{\xi\zeta\rho}, t_{\xi\rho\zeta}, t_{\zeta\xi\rho}, t_{\zeta\rho\xi}, t_{\rho\xi\zeta}, t_{\rho\zeta\xi}$: entiers ;
/* *Valeurs minimisées pour la consommation énergétique* */
$Min\rho\xi, Min\zeta\xi, Min\rho\zeta\xi, Min\zeta\rho\xi$: décimaux
/* *Valeurs minimisées pour le coût financier* */
$Min\xi\zeta, Min\rho\zeta, Min\xi\rho\zeta, Min\rho\xi\zeta$: décimaux ;
/* *Valeurs minimisées pour la pollution de l'environnement* */
$Min\xi\rho, Min\zeta\rho, Min\xi\zeta\rho, Min\zeta\xi\rho$: décimaux ;

/* *1 énergie, 2 prix, 3 pollution* */
/* *Trouver la plus petite valeur de* $TMin\xi$ *minimisant la fonction* ζ *puis la fonction* ρ */
/* *Définition de Trouver_T_MinALL() dans l'algorithme 3* */
$(t_{\xi\zeta\rho}, Min\xi, Min\xi\zeta et Min\xi\zeta\rho) \leftarrow Trouver\_T\_MinALL(\xi, \zeta, \rho, TMin\xi, x)$;

/* *1 énergie, 2 pollution, 3 prix* */
/* *Trouver la plus petite valeur de* $TMin\xi$ *minimisant la fonction* ρ *puis la fonction* ζ */
$(t_{\xi\rho\zeta}, Min\xi, Min\xi\rho et Min\xi\rho\zeta) \leftarrow Trouver\_T\_MinALL(\xi, \rho, \zeta, TMin\xi, x)$;

/* *1 prix, 2 énergie, 3 pollution* */
/* *Trouver la plus petite valeur de* $TMin\zeta$ *minimisant la fonction* ξ *puis la fonction* ρ */
$(t_{\zeta\xi\rho}, Min\zeta, Min\zeta\xi et Min\zeta\xi\rho) \leftarrow Trouver\_T\_MinALL(\zeta, \xi, \rho, TMin\zeta, y)$;

/* *1 prix, 2 pollution, 3 énergie* */
/* *Trouver la plus petite valeur de* $TMin\zeta$ *minimisant la fonction* ρ *puis la fonction* ξ */
$(t_{\zeta\rho\xi}, Min\zeta, Min\zeta\rho et Min\zeta\rho\xi) \leftarrow Trouver\_T\_MinALL(\zeta, \rho, \xi, TMin\zeta, y)$;

/* *1 pollution, 2 énergie, 3 prix* */
/* *Trouver la plus petite valeur de* $TMin\rho$ *minimisant la fonction* ξ *puis la fonction* ζ */
$(t_{\rho\xi\zeta}, Min\rho, Min\rho\xi et Min\rho\xi\zeta) \leftarrow Trouver\_T\_MinALL(\rho, \xi, \zeta, TMin\rho, z)$;

/* *1 pollution, 2 prix, 3 énergie* */
/* *Trouver la plus petite valeur de* $TMin\rho$ *minimisant la fonction* ζ *puis la fonction* ξ */
$(t_{\rho\zeta\xi}, Min\rho, Min\rho\zeta et Min\rho\zeta\xi) \leftarrow Trouver\_T\_MinALL(\rho, \zeta, \xi, TMin\rho, z)$;

return $t_{\xi\zeta\rho}$, $Min\xi$, $Min\xi\zeta$ et $Min\xi\zeta\rho$,
$t_{\xi\rho\zeta}$, $Min\xi$, $Min\xi\rho$ et $Min\xi\rho\zeta$,
$t_{\zeta\xi\rho}$, $Min\zeta$, $Min\zeta\xi$ et $Min\zeta\xi\rho$,
$t_{\zeta\rho\xi}$, $Min\zeta$, $Min\zeta\rho$ et $Min\zeta\rho\xi$,
$t_{\rho\xi\zeta}$, $Min\rho$, $Min\rho\xi$ et $Min\rho\xi\zeta$,
$t_{\rho\zeta\xi}$, $Min\rho$, $Min\rho\zeta$ et $Min\rho\zeta\xi$;

Algorithm 2 Définition de la fonction *Obtenir_T_Min*

```
Obtenir toutes les valeurs de t minimisant la fonction F
function OBTENIR_T_MIN(F {tableau de décimaux}, t {tableau d'entiers}, taille {taille du tableau t})

    MinF ← ∞ : décimal ; valeur minimum de F
    m : entier ; /* nombre de t minimisant F */
    TMinF[] : tableau d'entiers ; /* tableau des t minimisant F */
    for i ∈ [1..taille] do
        if F[t[i]] <= MinF then
            if F[t[i]] < MinF then
                m ← 0 ; /* car nouveau minimum */
                MinF ← F[t[i]] ; /* car nouveau minimum */
            end if
            m++
            TMinF[m] ← t[i] ;
        end if
    end for
    return (MinF, TMinF[ ], m) ;
end function
```

La simulation est considérée sur une période de 30 jours. Le temps est discrétisé en intervalles de temps égaux. Chaque intervalle de temps dure ΔT minutes. Dans notre cas, nous utilisons un ΔT égal à 15 minutes. Nous rappelons que la puissance électrique moyenne d'un noeud i pendant la réservation est égale à $\alpha_i P^i_{max}$. α_i est généré aléatoirement entre $\frac{P^i_{repos}}{P^i_{max}}$ et 1.

Le simulateur prend comme paramètres d'entrée :

- Une liste de puissances électriques au repos P_{repos} et maximales P_{max} des N_{Tot} noeuds du supercalculateur. N_{Tot} est le nombre total des noeuds du supercalculateur. Les puissances électriques P_{repos} et P_{max} de chaque noeud sont obtenues à partir des puissances électriques mesurées des 44 noeuds de calcul de la grappe *Stremi* de la plate-forme Grid'5000 (voir section 3.1). Pour passer à l'échelle exaflopique, nos simulations doivent intégrer un nombre de noeuds de l'ordre de 10^6. Pour avoir des valeurs réalistes par rapport à l'échelle exaflopique, nous avons considéré que chacun des noeuds avait une efficacité énergétique de 50 gigaFLOPS/W (1 exaFLOPS pour 20 MW comme préconise le DARPA [Bergman et al., 2008]). Par conséquent, nous avons pris $1 \cdot 10^6$ noeuds et considéré que la puissance électrique d'un noeud du supercalculateur est de 1/10e de la puissance électrique d'un noeud de la grappe *Stremi* et que chaque noeud du supercalculateur simulé est capable d'attendre 5 teraFLOPS. Ceci permet de constituer un supercalculateur de 5 exaFLOPS.
- L'agenda du coût financier. L'agenda que nous considérons est l'agenda réel des prix par joule facturés par EDF pendant le mois de janvier 2013 [2]. La

2. http://particuliers.edf.com/gestion-de-mon-contrat/options-tempo-et-ejp/option-tempo/l-historique-52426.html

Algorithm 3 Définition de la fonction $Trouver\_T\_MinALL$

/ Trouver la plus petite valeur de TMinF minimisant la fonction G puis la fonction H */*
/ Pour cela, on cherche tous les TMinF qui minimisent G. Parmi les temps restants, on cherche ceux qui minimisent H */*

function TROUVER_T_MINALL(F, G, H {tableau de décimaux}, TMinF {tableaux d'entiers minimisant F}, tailleF {taille du tableau }))

$TMinFG[]$: tableau d'entiers ; */* tableau des TMinF minimisant F puis G */*
$TMinFGH[]$: tableau d'entiers ; */* tableau des TMinF minimisant F, G puis H */*
$tailleFG, tailleFGH$: entiers ; */* tailles des tableaux TMinFG[] et TMinFGH[] */*
$MinFG$: décimal ; */* correspond à la valeur minimale de G après avoir minimisé F */*
$MinFGH$: décimal ; */* correspond à la valeur minimale de H après avoir minimisé F puis G */*

t_{FGH} : *entier* ; */* correspond à la plus petite valeur qui minimise F, G puis H */*

/ Choisir parmi TMinF toutes les valeurs donnant la plus petite valeur pour G */*
$(MinFG, TMinFG, tailleFG) \leftarrow Obtenir\_T\_Min(G, TMinF, tailleF)$;

/ Choisir parmi TMinFG toutes les valeurs donnant la plus petite valeur pour H */*
$(MinFGH, TMinFGH, tailleFGH) \leftarrow Obtenir\_T\_Min(H, TMinFG, tFG)$;

/ Choisir parmi TMinFGH la plus petite valeur (le plus tôt dans notre cas) */*
$t_{FGH} \leftarrow min(TMinFGH)$;
return t_{FGH}, $MinF$, $MinFG$ et $MinFGH$;
end function

figure 6.5 montre l'évolution du prix de l'électricité (en euros/J) sur le mois considéré et avec une granularité de ΔT (15 minutes). Pendant ce mois-ci, le prix du joule a varié entre 0.0212×10^{-6} et 0.1422×10^{-6} euros [3]. Les valeurs affichées en euros par joule ont été converties à partir des valeurs données en euros par kWh (voir axe des ordonnées principal à gauche).

– L'agenda de l'impact environnemental. L'agenda que nous considérons est l'agenda réel des quantités de CO_2 émises par joule en France pendant le mois de janvier 2013 [4]. La figure 6.5 montre également l'évolution de la quantité de CO_2 émise par joule sur le mois considéré et avec une granularité de ΔT (15 minutes). Pendant ce mois-ci, les quantités de CO_2 émises par joule ont varié entre 7.5 et 31.1 μg. Les valeurs affichées en μg par joule ont été converties à partir des valeurs données en $gparkWh$ (voir axe des ordonnées secondaire à droite).
– L'agenda de la puissance électrique maximale que le fournisseur d'énergie peut délivrer. Pour chaque intervalle de temps, la puissance électrique maximale qui

3. http://particuliers.edf.com/gestion-de-mon-contrat/options-tempo-et-ejp/option-tempo/details-de-l-option-52429.html
4. http://www.rte-france.com/fr/developpement-durable/eco2mix/emission-de-co2-par-kwh-d-electricite-produite-en-france

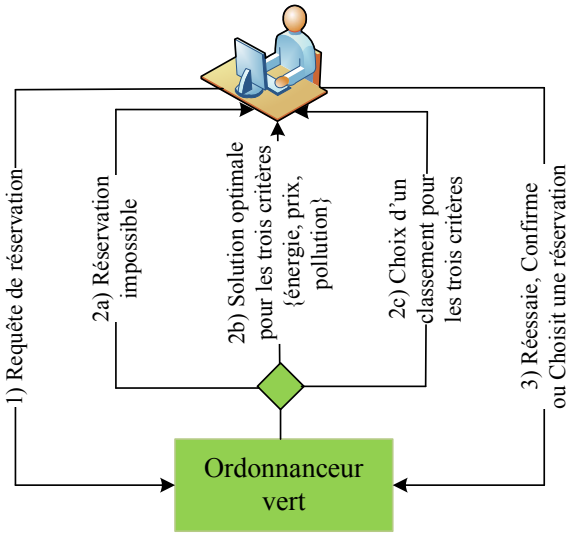

FIGURE 6.4 – Communications entre l'utilisateur et SESAMES pendant une réservation de ressources

peut être délivrée au supercalculateur est générée aléatoirement entre 0% et 200% de la puissance électrique maximale instantanée de tout le supercalculateur ($\sum_{i=1}^{N_{Tot}} P^i_{max}$). Ces valeurs sont générées selon une loi de probabilité normale avec une espérance égale à 100 et une variance égale à 25.

De plus, le simulateur prend en entrée un flux de R requêtes de réservations. Chaque requête de réservation r est générée aléatoirement et est composée de :

- Un nombre de noeuds N^r généré aléatoirement entre 1 et N_{Tot} noeuds.
- Une durée D^r générée entre 15 minutes (*i.e.,* la plus petite durée possible dans notre agenda simulé, à cause du ΔT) et une durée maximale correspondant à 24 heures. Nous limitons la durée maximale à 24 heures afin d'éviter de générer aléatoirement des durées de réservations longues par rapport aux 30 jours de l'agenda simulé. En effet, si des réservations de durée assez longues sont placées dans l'agenda, les futures requêtes de réservations de ressources risquent d'être plus souvent rejetées.
- Une date de début au plus tôt $t^r_{d_{min}}$ générée aléatoirement entre le premier et le dernier intervalle de temps possible sur toute la période de la simulation.
- Une date de début au plus tard $t^r_{d_{max}}$ égale à $t^r_{min} + T^r$ où T^r est le décalage maximal autorisé de la date de début de la réservation. Dans notre cas, T_r est égale à deux fois la durée D^r.

Après chaque requête de réservation, le simulateur maintient et met à jour trois agendas :

- L'agenda des réservations placées en utilisant l'approche d'ordonnancement qui consiste à démarrer les réservations dès que possible dans l'intervalle $[t^r_{d_{min}}; t^r_{d_{max}}]$ et à prendre les N^r premiers noeuds disponibles.

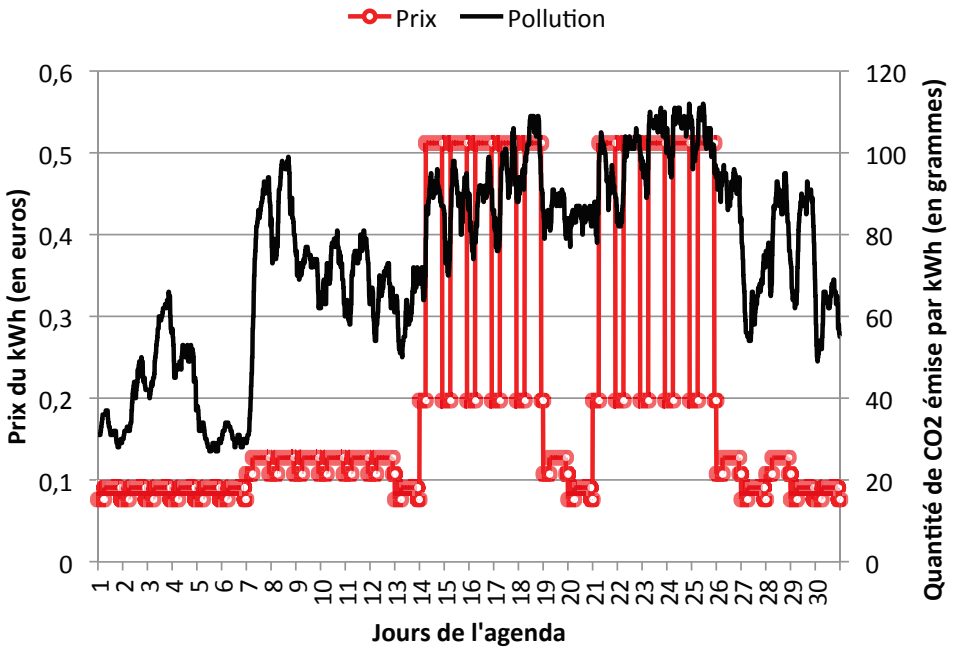

FIGURE 6.5 – Agendas du prix du kWh et de la quantité de CO_2 émise par kWh (données extraites des sites d'EDF et de RTE)

- L'agenda des réservations placées en utilisant notre approche d'ordonnancement avec un utilisateur qui classe les critères comme suit : coût financier, impact environnemental, puis consommation énergétique ;
- L'agenda des réservations placées en utilisant notre approche d'ordonnancement avec un utilisateur qui classe les critères comme suit : impact environnemental, coût financier puis consommation énergétique.

Afin de comparer les trois politiques d'ordonnancement, au fur et à mesure que des réservations sont planifiées, nous recueillons de chaque agenda de réservations les valeurs moyennes suivantes qui sont calculées sur l'ensemble des réservations déjà placées :

- la consommation moyenne d'énergie d'une réservation ;
- le coût financier moyen d'une réservation ;
- la quantité moyenne de CO_2 émis lors d'une réservation ;
- le décalage temporel moyen des dates de début des réservations par rapport aux dates de début au plus tôt de chaque demande de réservation ;
- le nombre de requêtes de réservations non satisfaites soit à cause de l'indisponibilité des noeuds soit parce que la puissance électrique maximale délivrée est insuffisante ;
- le taux moyen d'occupation de la plate-forme, c'est-à-dire la proportion moyenne de noeuds réservés calculée sur la durée totale de l'agenda des réservations.

Nous considérons que dans lors d'une simulation, il y a 100 requêtes de réserva-

tions de ressources qui sont statiques : elles arrivent toutes avant le premier jour de l'agenda et elles sont dans un ordre donné. Dans notre simulation, le choix de 100 requêtes de réservations est motivée par une volonté d'avoir un taux moyen d'occupation qui soit proche de 50 %. Il est à noter que si le taux d'occupation est de 100 %, il n'y aura aucune différence (ni énergétique, ni financière, ni environnementale) entre les trois approches comparées du fait que tous les noeuds seront pris à tous les moments de l'agenda. Les paramètres fixés dans notre simulateur sont résumés dans le tableau 6.1.

Nombre de jours	30
Taille de chaque intervalle de temps (ΔT)	15 minutes
Nombre total de noeuds (N_{tot}) du supercalculateur	1×10^6
Listes des P_{repos} et des P_{max} pour les N_{tot} noeuds	ceux de la grappe *Stremi*
Coefficient α_i déterminant la puissance moyenne du noeud i	généré aléatoirement entre $\frac{P^i_{repos}}{P^i_{max}}$ et 1
Nombre de réservations (R)	100
Arrivée des réservations	statique et ordonnée
Agenda des prix par joule	agenda fourni par EDF pour le mois de janvier 2013
Agenda des émissions de CO_2 par joule	agenda fourni par RTE France pour le mois de janvier 2013
Puissance électrique maximale ($P_{capping}$) délivrée	générée entre 0 % et 200 % de $\sum_{i=1}^{N_{Tot}} P^i_{max}$ selon la loi normale N(100, 25)
Nombre de noeuds (N^r) demandés lors de la requête r	généré aléatoirement entre 1 et N_{Tot} noeuds
Durée (D^r) d'une réservation	générée entre 15 minutes et 24 heures
Date ($t^r_{d_{min}}$) de début r au plus tôt de la réservation	générée aléatoirement sur toute la durée de l'agenda
Décalage temporel maximal du début de la réservation r (T^r)	$2 \times D^r$

TABLE 6.1 – Paramètres de simulation du flux de réservations à planifier

D'une simulation à l'autre, les valeurs obtenues peuvent être différentes car la plupart des paramètres de simulation sont tirés aléatoirement. Afin de lisser ce caractère aléatoire, nous exécutons cette simulation 100 fois et nous calculons les valeurs moyennes que nous obtenons sur l'ensemble des 100 simulations.

6.3.2 Résultats d'évaluation

La figure 6.6 présente l'énergie moyenne consommée sur toutes les réservations déjà placées. La figure 6.7 présente le coût financier moyen sur toutes les réservations déjà placées. La figure 6.8 présente la quantité de CO_2 émise sur toutes les réservations déjà placées. La figure 6.9 présente le décalage temporel moyen sur toutes les réservations déjà placées. Les réservations rejetées ne sont pas comptabilisées dans le calcul des résultats affichés.

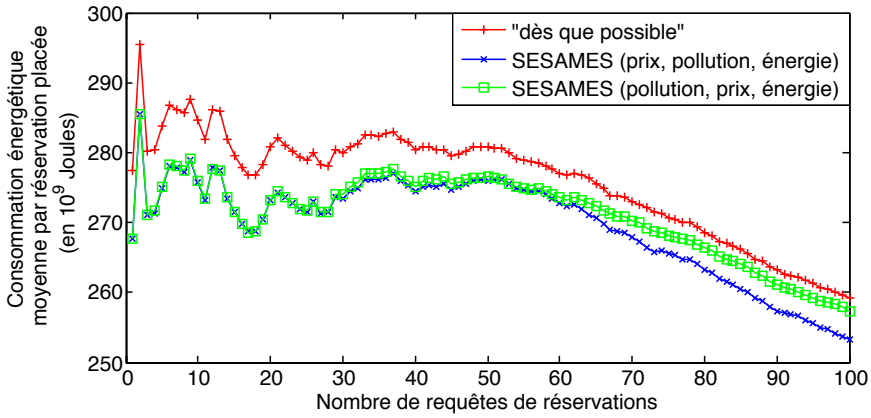

FIGURE 6.6 – Evolution moyenne de la consommation énergétique par réservation placée

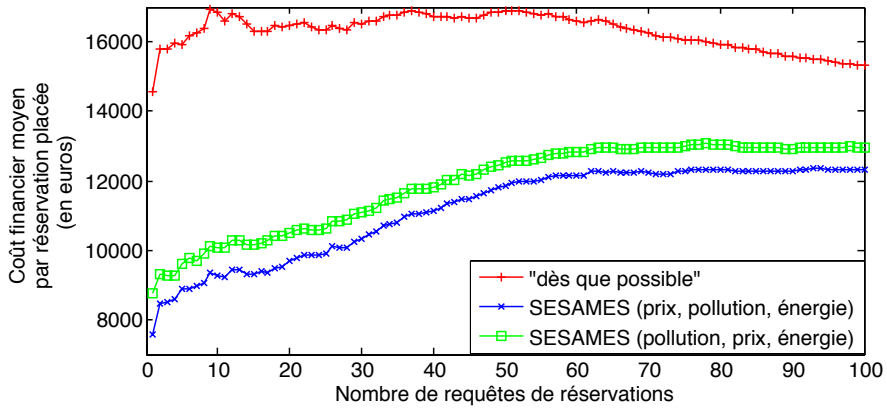

FIGURE 6.7 – Evolution moyenne du coût financier par réservation placée

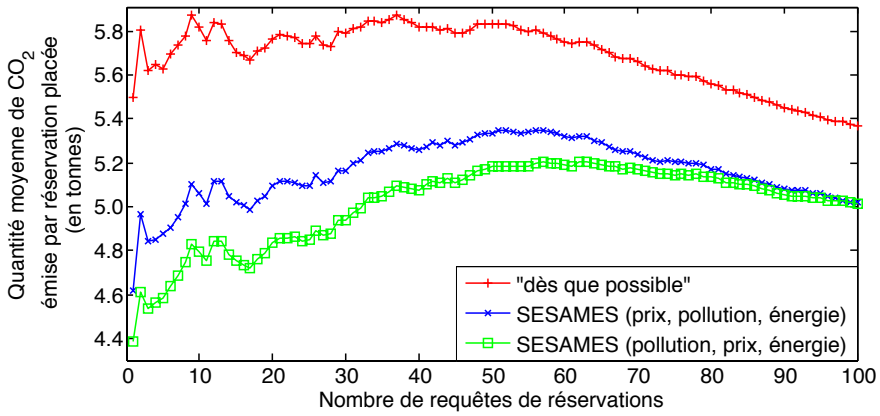

FIGURE 6.8 – Evolution moyenne des quantités de CO_2 émises par réservation placée

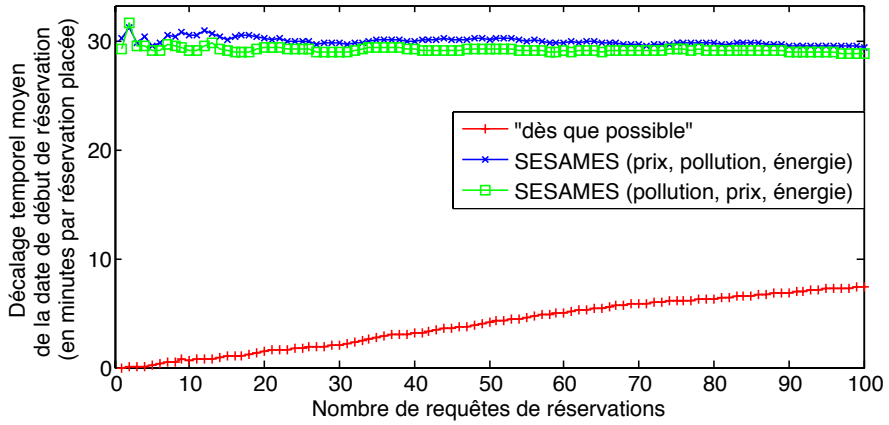

FIGURE 6.9 – Evolution moyenne du décalage temporel moyen de la date de début d'une réservation

Premièrement, nous remarquons sur les figures 6.6, 6.7 et 6.8 que l'approche d'ordonnancement multi-critères (avec un classement donné pour les trois critères) consomme moins d'énergie, coûte moins d'argent et génère moins de CO_2. En effet, en comparant les deux versions de l'approche d'ordonnancement multi-critères à l'approche "dès que possible", les économies relatives que nous obtenons après 100 requêtes de réservations sont exposées dans le tableau 6.2.

Les valeurs affichées dans les figures 6.6, 6.7 et 6.8 semblent être élevées. Ci-dessous, nous essayons de retrouver les ordres de grandeur de ces valeurs en se basant sur les paramètres moyens. En moyenne, un utilisateur demande la moitié de la plate-forme (5×10^5 noeuds) pendant une durée de réservation moyenne d'une demi-journée (*i.e.*, $12 \times 3600 = 43200$ s). En moyenne, un noeud consomme environ 17.5W

	SESAMES (prix, pollution, énergie)	SESAMES (pollution, prix, énergie)
Énergie économisée	2.32 %	0.7 %
Argent économisé	24.22 %	18.23 %
Émissions CO_2	6.98 %	7.12 %

TABLE 6.2 – Synthèse des économies relatives par rapport à l'approche "dès que possible"

($0.75 \times 25W$, avec un α_i égal à 0.8). Par conséquent, la consommation énergétique moyenne par réservation est de $(5 \times 10^5 noeuds) \times 17.5W \times 43200s$. Ainsi, son ordre de grandeur est de $3 \cdot 10^{11}$ joules (ce qui est le même ordre de grandeur que la figure 6.6). En moyenne le prix de l'électricité est égal à 5×10^{-8} euros par joule. Par conséquent, l'ordre de grandeur du coût financier moyen par réservation est $(3 \times 10^{11}) \times (5 \times 10^{-8})$ ce qui est égal à 1.5×10^4 (ce qui est le même ordre de grandeur que la figure 6.7). De façon similaire, lorsque nous multiplions l'énergie moyenne consommée par réservation par la quantité de CO_2 émise par joule, nous obtenons le même ordre de grandeur que celui que nous obtenons pour la quantité moyenne de CO_2 émise par réservation.

Au fur et à mesure que le nombre de réservations planifiées croît, nous remarquons que le coût financier moyen (figure 6.7) et la quantité moyenne de CO_2 émise (figure 6.8) tendent à décroître pour les deux versions de l'approche d'ordonnancement multi-critères. En effet, plus le nombre de réservations planifiées croît, plus les intervalles de temps (où les noeuds disponibles consomment le moins, où l'énergie coûte le moins cher et où elle génère le moins de pollution) diminuent. En outre, pour les deux versions de l'approche d'ordonnancement multi-critères, nous remarquons que le décalage temporel moyen du début de réservation se stabilise à 30 minutes par réservation alors que le décalage maximal paramétré est en moyenne de 2 fois une demi-journée (soit 24 heures).

Par ailleurs, pour l'approche d'ordonnancement "dès que possible", nous remarquons que le décalage temporel moyen par réservation croît au fur et à mesure que le nombre de réservations planifiées augmente. En effet, étant donné que l'agenda des réservations avec l'approche "dès que possible" se remplit au fur et à mesure, alors il devient de plus en plus difficile de placer des réservations avec une date de début qui n'est pas décalée dans le temps.

Nous constatons également que l'énergie consommée pour les trois politiques d'ordonnancement tend à diminuer plus le nombre de réservations planifiées augmente. Étant donné que les agendas de réservation se remplissent, il devient de plus en plus difficile de placer des réservations avec un grand nombre de noeuds. Les réservations qui demandent un grand nombre de noeuds sont rejetées. Par conséquent, au fur et à mesure que l'agenda se remplit, les réservations placées sont celles qui demandent de moins en moins de noeuds. Ceci implique que l'énergie moyenne par réservation décroît au fur et à mesure que l'agenda se remplit (*i.e.,* que la simulation

avance).

Le taux d'occupation de la plate-forme et le taux des réservations non placées après traitement des 100 requêtes de réservations, sont donnés dans le tableau 6.3. Une réservation peut être non placée car elle peut être rejetée soit parce que le nombre de noeuds demandés n'est pas disponible soit parce que cette réservation engendrerait un dépassement de la puissance électrique maximale délivrée par le fournisseur d'énergie.

	Taux d'occupation	Réservations non placées à cause du nombre de noeuds	Réservations non placées à cause de $P_{capping}$
"Dès que possible"	52.59 %	7.56 %	3.52 %
SESAMES (prix, pollution, énergie)	51.60 %	7.12 %	4.02 %
SESAMES (pollution, prix, énergie)	52.29 %	8.16 %	3.41 %

TABLE 6.3 – Taux d'occupation de la plate-forme et taux des réservations non placées.

Pour les trois politiques d'ordonnancement, nous constatons bien que plus que la moitié de la plate-forme est réservée après traitement des 100 requêtes de réservations. Les deux versions de l'approche d'ordonnancement proposée par SESAMES génèrent des taux de réservations non placées très voisins de ceux de l'approche "dès que possible".

Nous remarquons qu'une partie non négligeable des réservations non placées est due à la limitation de la puissance électrique maximale délivrée. Par exemple, il est de 31% du taux total des réservations non placées pour la politique d'ordonnancement "dès que possible", de 36% pour "SESAMES (prix, pollution, énergie)", et 30 % pour "SESAMES (pollution, prix, énergie)". Il est donc important de tenir compte de cette limitation de la puissance électrique maximale pour placer de nouvelles réservations dans l'agenda. Une approche d'ordonnancement qui n'en tiendrait pas compte, placerait la réservation dans l'agenda et cela engendrerait des coupures d'électricité pendant la réservation. Ces coupures d'électricité auraient lieu aux moments où l'application de l'utilisateur dépasse cette puissance maximale.

6.4 Conclusions du chapitre

Dans ce chapitre, nous avons présenté SESAMES, un environnement logiciel dont le but est de gérer l'exécution des applications sur des plates-formes de calcul très

haute performance dans une perspective de consommer moins et mieux. Nous avons pu présenter les différentes fonctionnalités de SESAMES, ses différents composants, ses interactions avec les multiples acteurs de la plate-forme de calcul très haute performance.

Afin de "consommer mieux", SESAMES s'appuie notamment sur une grille électrique intelligente qui lui permet d'interagir efficacement avec le fournisseur d'énergie. En mettant en place un dialogue avec le fournisseur d'énergie, SESAMES échange des informations avec celui-ci et récupère principalement des données relatives aux tarifs de facturation de l'énergie, aux émissions de CO_2 issues des différentes sources d'énergie utilisées pour la production de l'électricité et aux limitations en termes de puissance électrique crête.

En prenant en compte ces informations et en s'appuyant sur une calibration énergétique de la plate-forme de calcul très haute performance, SESAMES offre une approche efficace en énergie et multi-critères pour planifier les réservations de ressources qu'expriment les différents utilisateurs de la plate-forme. Des simulations montrent que SESAMES permet de réduire à la fois la consommation énergétique, le coût financier et l'impact environnemental liés à une réservation de ressources, tout en respectant les contraintes imposées par l'utilisateur et le fournisseur d'énergie.

Si le propriétaire de la plate-forme de calcul très haute performance facture l'utilisateur en fonction de la consommation énergétique, du coût financier, ou/et des émissions de CO_2 liés à sa réservation de ressources, SESAMES permettra à l'utilisateur de réduire sa facture auprès du propriétaire de la plate-forme. Afin que l'approche proposée par SESAMES soit financièrement plus profitable au propriétaire de la plate-forme, ce dernier devrait éteindre les ressources inutilisés tout en veillant à ce que la durée de l'extinction de ces ressources soit supérieure à un certain seuil minimal [Orgerie et al., 2008b].

7 Conclusions et perspectives

La question de l'efficacité énergétique est un enjeu majeur dans les infrastructures de calcul très haute performance. En effet, le supercalculateur qui atteint la plus grande puissance de calcul (34 pétaFLOPS) selon le classement du TOP 500 publié en juin 2013, consomme déjà plus de 17 MW alors que le DARPA fixe à 20 MW la puissance électrique maximale à ne pas dépasser pour une machine exaflopique (1 exaFLOPS soit 1000 pétaFLOPS). Dans cette thèse, nous nous sommes intéressés à proposer des solutions afin que la consommation énergétique dans ces infrastructures soit à la fois réduite et meilleure. D'une part, nous proposons d'utiliser les noeuds de calcul et les versions des services applicatifs qui permettent de consommer moins d'énergie. Les services applicatifs que nous avons considérés dans notre étude sont les protocoles de tolérance aux pannes et les algorithmes de diffusion de données. En ce qui concerne la tolérance aux pannes, nous nous sommes focalisés sur la phase de fonctionnement sans panne en considérant d'une part le cas des protocoles coordonnés et d'autre part le cas des protocoles non coordonnés. En ce qui concerne la diffusion de données, nous avons considéré deux algorithmes utilisés dans MPI (*MPI/SAG* et *MPI/Pipeline*) et deux algorithmes hybrides combinant chacun des deux précédents avec OpenMP (respectivement *Hybrid/SAG* et *Hybrid/Pipeline*). D'autre part, nous proposons de planifier les réservations de ressources tout en visant à réduire la consommation énergétique, le coût financier et la pollution de l'environnement.

7.1 Conclusions

Dans cette thèse, nous avons d'abord étudié les différents travaux existants qui visent à évaluer et à consommer efficacement l'énergie dans les infrastructures de calcul très haute performance. Bien que ces travaux ne répondent pas à tous les besoins que nous avons identifiés dans le chapitre 1, ils révèlent que la recherche

scientifique visant à améliorer la consommation énergétique dans ces infrastructures a été très active lors de ces dernières années.

Dans un premier temps, nous avons essayé de comprendre le comportement électrique des deux services applicatifs étudiés. Pour cela, nous avons essayé de mesurer la puissance électrique de quelques opérations que nous identifions dans les protocoles de tolérance aux pannes et les algorithmes de diffusion de données. Suite à nos premières mesures de la puissance électrique, nous avons montré qu'il est important que l'équipement de mesure soit suffisamment précis pour que les mesures relevées soient fiables. Par ailleurs, nous avons noté que les machines issues d'une même grappe de calcul ne consomment pas forcément la même puissance électrique bien qu'elles soient identiques d'un point de vue matériel et qu'elles exécutent les mêmes opérations. D'une part, l'hétérogénéité électrique des noeuds identiques d'une même grappe est due uniquement aux différences de puissance électrique de ces noeuds au repos. D'autre part, le surcoût dynamique de la puissance électrique liée à l'exécution d'une opération donnée est homogène pour des noeuds identiques issus d'une même grappe. Une méthodologie expérimentale nous a permis de détecter que cette hétérogénéité de la puissance électrique provenait principalement des processeurs et des ventilateurs chargés de les refroidir.

En plus d'être difficile, la mesure de la consommation énergétique peut être couteuse en temps car elle nécessite d'exécuter l'application ou le service applicatif à très grande échelle et dans tous les contextes d'exécution. Afin de réduire le nombre de mesures et de connaître la consommation énergétique du service applicatif sans pré-exécuter l'application, nous avons présenté une méthodologie pour estimer de façon précise la consommation énergétique des opérations identifiées dans les services applicatifs, et ce pour n'importe quel contexte d'exécution et pour n'importe quelle plate-forme dotée de wattmètres pour chaque ressource (serveurs, stockage, équipements réseaux). Nous pouvons imaginer que les futurs unités de distribution électrique (PDUs) permettront de mesurer la puissance électrique avec une fréquence plus élevée et avec plus de précision que les équipements actuellement disponibles.

Notre méthodologie d'estimation repose sur une calibration de la plate-forme d'exécution et une description du contexte d'exécution qui consiste à fournir un ensemble de paramètres en provenance de la plate-forme et de l'application. Nous appelons calibration de la plate-forme, le processus qui consiste à orchestrer la collecte d'un ensemble de mesures de puissance électrique et de temps d'exécution des différentes opérations identifiées dans l'optique d'adapter les estimations énergétiques en fonction de la plate-forme. En comparant les estimations énergétiques de chacune des opérations identifiées aux mesures énergétiques de ces opérations lors de l'exécution de l'application, nous avons pu montrer que les écarts étaient relativement faibles. Nos estimations énergétiques sont donc précises pour chacune des opérations que ce soit pour la tolérance aux pannes ou pour la diffusion de données. En nous appuyant sur les estimations énergétiques des différentes opérations, nous sommes capables d'évaluer la consommation énergétique d'un protocole de tolérance aux pannes ou d'un algorithme de diffusion de données.

Une des solutions que nous proposons pour réduire la consommation énergétique de ces deux services applicatifs, est de fournir à l'utilisateur les estimations énergétiques avant l'exécution de l'application afin de lui offrir la possibilité de choisir le protocole ou l'algorithme qui consomme le moins d'énergie en fonction du contexte d'exécution et de l'application envisagés. En exploitant les différences de puissance électrique qui caractérisent les noeuds identiques d'une même grappe, l'autre solution pour consommer moins d'énergie consiste à privilégier l'utilisation des noeuds de calcul dont la puissance électrique au repos est la moindre. D'une part, nous avons montré qu'en choisissant d'exécuter une application sur les noeuds qui consomment le moins, l'utilisateur peut réduire significativement la consommation énergétique de son application. Ainsi, s'il est facturé en fonction de sa consommation énergétique, cette approche lui permettra d'économiser de l'énergie et donc de gagner de l'argent. D'autre part, nous avons montré qu'en attribuant à l'utilisateur les noeuds qui consomment le moins, le propriétaire de la plate-forme peut économiser plus d'électricité s'il décide d'éteindre les noeuds restants inutilisés, c'est-à-dire les noeuds qui consomment le plus. Pour que cette approche soit financièrement profitable au propriétaire de l'infrastructure, il faut que celui-ci facture l'utilisateur en fonction du nombre de noeuds.

En plus de la réduction de la consommation énergétique, nous proposons d'intégrer deux nouveaux critères afin de "consommer mieux" : le coût financier et la pollution de l'environnement (quantité de CO_2 émis). À ce titre, nous avons proposé SESAMES, un environnement logiciel qui s'appuie sur une grille électrique intelligente pour interagir efficacement avec le fournisseur d'énergie. SESAMES échange des informations avec ce dernier et récupère principalement des données relatives aux tarifs de facturation de l'énergie, aux émissions de CO_2 dues aux différentes sources d'énergie utilisées pour la production de l'électricité et les limitations en termes de puissance électrique crête. En prenant en compte ces informations et en s'appuyant sur une calibration énergétique de la plate-forme de calcul très haute performance, SESAMES permet d'ordonnancer les réservations de ressources qu'expriment les différents utilisateurs de la plate-forme, en réduisant à la fois la consommation énergétique, le coût financier et l'impact environnemental, tout en respectant les contraintes imposées par l'utilisateur et le fournisseur d'énergie.

7.2 Discussion et Perspectives

Dans cette section, nous discutons certaines de nos conclusions et identifions plusieurs questions ouvertes qui pourraient constituer des pistes pour des travaux futurs.

7.2.1 Compromis entre la consommation énergétique et le temps d'exécution des services applicatifs

Dans cette thèse, nous avons mis l'accent sur la réduction de la consommation énergétique des services applicatifs en nous appuyant sur les estimations énergétiques des différentes opérations que nous identifions. Étant donné que nous sommes également capables d'estimer le temps d'exécution lié à ces services applicatifs, nous pouvons imaginer que nous fournissons également de telles estimations aux utilisateurs afin de leur permettre de faire un choix de service applicatif en tenant compte simultanément des deux critères : la consommation énergétique et le temps d'exécution.

On pourrait penser que le protocole de tolérance aux pannes ou l'algorithme de diffusion de données qui dure le moins longtemps est forcément celui qui consomme le moins d'énergie. Ceci n'est pas toujours vrai. Bien que la coordination puisse durer moins longtemps que l'enregistrement de messages sur disque dur, le fait que la puissance électrique liée à la coordination soit bien plus importante que celle de l'enregistrement de messages sur disque dur peut impliquer que la consommation énergétique de la coordination soit plus grande que celle de l'enregistrement de messages sur disque dur, et ce même dans le cas où leurs temps d'exécution respectifs ont des tendances inverses. Donc, le protocole coordonné bien que prenant moins de temps, peut dans certains cas consommer plus d'énergie que le protocole non coordonné.

Nous avons également constaté cela dans le cas des algorithmes de diffusion de données. En effet, bien que les consommations énergétiques des algorithmes hybrides peuvent être plus basses que celles des algorithmes utilisés dans MPI dans les applications, les temps d'exécution des algorithmes hybrides peuvent être légèrement plus élevés que ceux des algorithmes utilisés dans MPI. Ceci est dû au fait que la puissance électrique des algorithmes hybrides pendant les opérations MPI (*Scatter*, *AllGather* et *Pipeline*) est beaucoup plus basse que celle des algorithmes utilisés dans MPI (un seul processus est actif dans le cas des algorithmes hybrides *vs* tous les processus déployés sont actifs dans le cas des algorithmes utilisés dans MPI).

7.2.2 Inclusion des mécanismes de récupération des points de reprise

Dans notre étude, nous avons considéré les protocoles de tolérance aux pannes pendant le fonctionnement normal de l'application, c'est-à-dire lorsqu'il n'y a aucune panne. Comme extension de notre étude, nous pouvons envisager d'estimer également les opérations liées aux mécanismes de récupération des points de reprise qui sont déclenchés en cas de panne. Les différentes opérations que nous identifions sont :

– Restauration des points de reprise : cette opération consiste à charger en mémoire le dernier point de reprise sauvegardé. Contrairement aux protocoles de sauvegarde coordonnée des points de reprises, dans les protocoles non coordonnés, seuls les processus qui sont tombés en panne ont besoin de redémarrer.

– Ré-exécution de l'application : cette opération consiste à exécuter l'application entre le moment où le point de reprise est restauré et l'instant où était arrivée l'exécution de l'application avant la panne. Contrairement aux protocoles de sauvegarde coordonnée des points de reprises, dans les protocoles non coordonnés, seuls les processus qui sont tombés en panne ré-exécutent l'application depuis le dernier point de reprise.
– Attente en récupération : dans le cas du protocole non coordonné, les processus qui ne sont pas tombés ont panne envoient aux processus défaillants les messages enregistrés et se mettent à attendre activement les processus qui doivent procéder restaurer le dernier point de reprise et ré-exécuter l'application. Cette opération concerne donc seulement les processus qui ne sont pas tombés en panne. Cette attente en récupération dure donc autant que la restauration et la ré-exécution de l'application qui sont effectuées par les processus défaillants.

En estimant la consommation énergétique de ces différentes opérations, nous serons capables de comparer les consommations énergétiques des protocoles coordonnés et non coordonnés dans le contexte d'une ou plusieurs pannes. Ceci permettra à l'utilisateur de choisir le protocole de tolérance aux pannes qui consomme le moins d'énergie non seulement en fonction de l'intervalle de points de reprise mais aussi du nombre de pannes. Le nombre moyen de pannes peut être évalué grâce au calcul du MTBF (*Mean Time Between Failures*).

Ainsi, pour savoir quel protocole choisir dans le contexte de pannes, il faudra évaluer :

$$\begin{aligned}\Delta E &= E_{Coordonne} - E_{NonCoordonne} \\ &= C \cdot E_{coordination} - E_{enregistrement} \\ &+ \sum_{j=1}^{F}[\sum_{i=1}^{N-X_j}(t_{restauration} \cdot (\rho^i_{restauration} - \rho^i_{attente}) \\ &+ \frac{T}{2} \cdot (\rho^i_{appli} - \rho^i_{attente}))]\end{aligned}$$

C est le nombre de points de reprise sauvegardés pendant l'application. T est l'intervalle de prise des points de reprise. Par conséquent, une panne a lieu en moyenne à $\frac{T}{2}$ après la prise du point de reprise. C'est donc également le temps de ré-exécution de l'application. F est le nombre de pannes ayant lieu pendant l'application. N est le nombre de noeuds mis en jeu dans l'application. X_j est le nombre de noeuds qui sont défaillants lors de la panne j. $E_{coordination}$ est la consommation énergétique d'une coordination. $E_{enregistrement}$ est la consommation énergétique liée à l'enregistrement de tous les messages. $t_{restauration}$ est le temps nécessaire pour restaurer le dernier point de reprise. $\rho^i_{restauration}$ est la puissance électrique du noeud i lors de la restauration du dernier point de reprise. $\rho^i_{attente}$ est la puissance électrique du noeud i lors de l'attente en récupération. ρ^i_{appli} est la puissance électrique du noeud i lors

de la ré-exécution de l'application.

$\rho^i_{attente}$ est le même que celui que nous avons identifié lors de la coordination des processus qui a lieu juste avant la sauvegarde de points de reprise dans le cas du protocole coordonné. $\rho^i_{restauration}$ et $t_{restauration}$ peuvent être calibrés de façon analogue à la puissance électrique et au temps d'exécution de la sauvegarde de points de reprise. Une des perspectives de cette thèse concerne l'estimation de la consommation électrique de l'application avant son éxecution (ρ^i_{appli}).

7.2.3 Appliquer des leviers énergétiques visant à réduire la consommation énergétique

Nous avons pu relever que parmi les opérations liées aux protocoles de tolérance aux pannes, les processus passent par des phases d'attente active, et ce afin d'être plus réactifs. Dans ce cas, l'énergie est consommée inutilement. Le cas le plus intéressant est celui de la récupération non coordonnée des points de reprise. En effet, les processus qui ne sont pas tombés en panne restent en attente active pendant $t_{restauration} + \frac{T}{2}$ (en moyenne). Ce temps peut être assez long : plusieurs minutes. De plus, en supposant que le nombre de noeuds tombés en panne est très faible par rapport au nombre total de noeuds mis en jeu dans l'application, de nombreux processus resteront en attente active pendant un temps assez long dans le cas de la récupération non coordonnée de points de reprise.

Afin de réduire la consommation énergétique, une des perspectives futures de la thèse serait d'appliquer des leviers énergétiques verts pendant cette attente active. En effet, nous pouvons soit éteindre puis rallumer les noeuds (ou les coeurs de calcul) concernés, soit ralentir les composants matériels (coeurs de calcul, carte réseau, etc) inutilisés. Toutefois, il faut veiller à appliquer ces leviers énergétiques avec précaution. D'abord, il faut s'assurer que l'administrateur de la plate-forme de calcul très haute performance autorise d'appliquer de telles actions sur les ressources de calcul de l'infrastructure. De plus, il faut que la durée de l'attente active soit suffisamment longue pour que l'extinction puis le rallumage des ressources de calcul soient profitables d'un point de vue énergétique. En effet, pour que l'extinction puis le rallumage des ressources permette d'économiser de l'énergie, la durée de l'attente active doit être supérieure au seuil minimal T_s mis en évidence par [Orgerie et al., 2008b] (voir section 2.2.1.2). Dans le cas contraire, il faut privilégier le ralentissement des ressources de calcul, par exemple en utilisant la technique DVFS pour les processeurs. Par ailleurs, il faut également tenir compte du fait que ces deux techniques de réduction de la consommation énergétique peuvent influencer le taux de pannes comme le soulignent les auteurs de l'article [Aupy et al., 2012]. Nous pouvons également estimer les économies d'énergie que permettent de telles optimisations énergétiques.

Si nous considérons que l'on éteint les noeuds qui ne sont pas tombés en panne dans le cas de la récupération non coordonnée, et que donc $\rho^i_{attente}$ est nulle, l'équation de la section 7.2.2 devient :

$$\begin{aligned}\Delta E &= E_{Coordonne} - E_{NonCoordonne} \\ &= C \cdot E_{coordination} - E_{enregistrement} \\ &+ \sum_{j=1}^{F}[\sum_{i=1}^{N-X_j}(t_{restauration} \cdot \rho^i_{restauration} + \frac{T}{2} \cdot \rho^i_{appli})]\end{aligned}$$

Plus le nombre de pannes F croît, plus ΔE augmente. En effet, $C \cdot E_{coordination} - E_{enregistrement}$ ne dépend pas du nombre de pannes et $\sum_{j=1}^{F}[\sum_{i=1}^{N-X_j}(t_{restauration} \cdot \rho^i_{restauration} + \frac{T}{2} \cdot \rho^i_{appli})$ est positif et augmente avec le nombre de pannes. Ceci signifie que plus le nombre de pannes devient important, plus il sera intéressant de choisir le protocole non coordonné.

7.2.4 Vers un ordonnancement des réservations de ressources multi-objectifs optimisé qui accepte plus de requêtes

Dans notre étude, telle que nous l'avons présentée dans le chapitre 6, l'approche multi-objectifs d'ordonnancement des réservations de ressources est profitable financièrement du point de vue de l'utilisateur. Pour qu'elle soit encore plus profitable pour le propriétaire de la plate-forme, il doit éteindre les noeuds qui ne sont pas utilisés à condition que le temps séparant deux réservations successives de ces noeuds soit suffisamment long. Si ce temps est relativement court (inférieur au T_s [Orgerie et al., 2008b]), il peut toujours prévoir de ralentir les noeuds de calcul inutilisés afin de les faire passer dans des états de basse consommation énergétique. Afin de réduire les coûts liées à l'extinction et au rallumage des ressources, il est plus convenable d'avoir à les éteindre le plus longtemps et le moins souvent [Orgerie et al., 2008a]. Dans cette optique, une des perspectives de cette thèse pourrait être d'intégrer ce critère permettant de réduire la consommation énergétique dans notre approche multi-objectifs d'ordonnancement. Pour cela, cette approche devrait chercher à "coller" les réservations entre elles de telle sorte qu'il y ait le minimum de "trous" dans l'agenda des réservations. De plus, pour être encore plus réalistes, les simulations pour valider une telle approche multi-objectifs pourraient intégrer une arrivée dynamique des requêtes de réservation (*i.e.,* au cours de la simulation).

Par ailleurs, nous avons considéré que les réservations des ressources de calcul n'étaient pas divisibles, c'est-à-dire que l'on ne pouvait pas arrêter une réservation pour la reprendre plus tard. Or, il est possible techniquement d'arrêter une application en sauvegardant son état dans un point de reprise et de reprendre son exécution en restaurant le point de reprise sauvegardé. Grâce à cette approche, une autre perspective de la thèse serait de considérer que les réservations de ressources sont divisibles. Ceci peut permettre de planifier une réservation de ressource sur plusieurs créneaux disjoints et donc d'exploiter encore mieux les créneaux où l'énergie est la moins chère et la moins polluante. Toutefois, il faut prendre en compte les coûts énergétiques, financiers et environnementaux liées à la sauvegarde et à la récupération des points de reprise.

Publications

Journaux internationaux avec comité de lecture

[1] Mohammed El Mehdi Diouri, Ghislain Landry Tsafack Chetsa, Olivier Glück, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf et Geoges Da Costa, Energy efficiency in HPC : with or without knowledge on applications and services, In International Journal of High-Performance Computing Applications (IJHPCA), vol. 27 no. 3 pp. 232-243, Aout 2013.

[2] Mohammed El Mehdi Diouri, Olivier Glück et Laurent Lefèvre, Smart Energy Management for Greener Supercomputing, In the Special Issue "Smart Energy Systems" of the European Research Consortium for Informatics and Mathematics News (ERCIM News), Number 92, Janvier 2013.

Conférences internationales avec actes et comité de lecture

[3] Mohammed El Mehdi Diouri, Olivier Glück, Laurent Lefèvre et Jean-Christophe Mignot, Energy Estimation for MPI Broadcasting Algorithms in Large Scale HPC Systems, In 20^{th} *European MPI Users Group Meeting on Recent Advances in Message Passing Interface (EuroMPI)*, Madrid, Spain, Septembre 2013.

[4] Mohammed El Mehdi Diouri, Olivier Glück et Laurent Lefèvre, SESAMES : a Smart-Grid Based Framework for Consuming Less and Better in Extreme-Scale Infrastructures, In 9^{th} *IEEE International Conference on Green Computing and Communications (GreenCom)*, Beijing, China, 20-23 Aout 2013.

[5] Mohammed El Mehdi Diouri, Olivier Glück, Laurent Lefèvre et Jean-Christophe Mignot, Your Cluster is not Power Homogeneous : Take Care when Designing Green Schedulers ! In 4^{th} *IEEE International Green Computing Conference (IGCC)*, Arlington, Virginia, USA, 27-29 Juin 2013

[6] Mohammed El Mehdi Diouri, Olivier Glück, Laurent Lefèvre et Franck Cappello, ECOFIT : A Framework to Estimate Energy Consumption of Fault Tolerance protocols during HPC executions, In 13^{th} *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Delft, Netherlands, 13-16 Mai 2013.

[7] Mohammed El Mehdi Diouri, Manuel F. Dolz, Olivier Glück, Laurent Lefèvre Pedro Alonso, Sandra Catalán, Rafael Mayo, et Enrique S. Quintana-Ortí, Solving some Mysteries in Power Monitoring of Servers : Take Care of your Wattmeters ! In *the International Conference on Energy-Efficiency in Large Scale Distributed Systems (EE-LSDS)*, Vienna, Austria, 22-24 Avril 2013.

[8] Mohammed El Mehdi Diouri, Olivier Glück, Laurent Lefèvre et Franck Cappello, Towards an Energy Estimator for Fault Tolerance Protocols, 18^{th} *ACM/SIGPLAN Symposium on Principles and Practice of Parallel Programming (poster paper), PPoPP2013*, Shenzhen, China, 23-27 Février 2013.

[9] Mohammed El Mehdi Diouri, Olivier Glück, Laurent Lefèvre et Franck Cappello, Energy considerations in Checkpointing and Fault tolerance protocols, In 2^{nd} *International Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS), co-located with the* 42^{th} *IEEE/IFIP International conference on Dependable Systems and Networks (DSN)*, Boston, Massachusetts, USA, 25-29 Juin 2012.

[10] Mohammed El Mehdi Diouri, Olivier Glück et Laurent Lefèvre, Towards a novel Smart and Energy-Aware Service-Oriented Manager for Extreme-Scale applications, In 1^{st} *International*

Workshop on Power-Friendly Grid Computing (PFGC), co-located with the 3^{rd} *IEEE International Green Computing Conference (IGCC)*, San José, California, USA, 5-9 Juin 2012.

Conférences nationales avec actes et comité de lecture

[11] Mohammed El Mehdi Diouri, Olivier Glück et Laurent Lefèvre, Vers des machines exaflopiques vertes, In 20^{e} *édition des Rencontres francophones du Parallélisme (RenPar)*, Saint-Malo, France, Mai 2011.

Divers

[12] Mohammed El Mehdi Diouri, Olivier Glück, Laurent Lefèvre, et Franck Cappello, Energy Evaluation in Checkpointing protocols, Papier court accepté et présenté dans le cadre de *l'école de recherche Grid'5000 (G5K school)*, Nantes, France, Décembre 2012.

[13] Mohammed El Mehdi Diouri et Laurent Lefèvre, Green Cloud and Energy Efficient Exascale Services, Poster au stand INRIA dans le cadre de la *SuperComputing conference (International conference for High performance computing, networking, storage and analysis), SC'2012*, Salt Lake City, Utah, USA, Novembre 2012.

[14] Mohammed El Mehdi Diouri, Olivier Glück, Laurent Lefèvre, et Olivier Mornard, Monitoring Internal Power Consumption of Computing Systems, Démonstration et poster présentés aux *Rencontres INRIA et Industrie (RII)*, Montbonnot, France, Mars 2012.

[15] Mohammed El Mehdi Diouri, Olivier Glück et Laurent Lefèvre, Green Networking for Exascale Systems, Papier court accepté et poster présenté à l'école d'été ResCom 2011, La Palmyre, France, Juin 2011.

[16] Marcos Dias de Assunçao, Mohammed El Mehdi Diouri, Laurent Lefèvre, Olivier Mornard, Anne-Cécile Orgerie et Ghislain Landry Tsafack Chetsa, Put some Green in your Grid'5000 experiments ! Travaux pratiques proposés dans le cadre de *l'école de recherche Grid'5000 (G5K school)*, Reims, France, Avril 2011.

References

Bibliographie

[Ali et al., 2012] Ali, S., Ahmad, R., and Kim, D.-H. (2012). A study of pricing policy for demand response of home appliances in smart grid based on m2m. In *10th International Conference on Frontiers of Information Technology, FIT 2012, Islamabad, Pakistan, December 17-19, 2012*, pages 231–236.

[Aloisio and Fiore, 2009] Aloisio, G. and Fiore, S. (2009). Towards Exascale Distributed Data Management. *IJHPCA*, 23(4) :398–400.

[Alonso et al., 2012] Alonso, P., Badia, R. M., Labarta, J., Barreda, M., Dolz, M. F., Mayo, R., Quintana-Ortí, E. S., and Reyes, R. (2012). Tools for power-energy modelling and analysis of parallel scientific applications. In *41st International Conference on Parallel Processing, ICPP 2012, Pittsburgh, PA, USA, September 10-13, 2012*, pages 420–429.

[Aupy et al., 2012] Aupy, G., Benoit, A., and Robert, Y. (2012). Energy-aware scheduling under reliability and makespan constraints. In *19th International Conference on High Performance Computing, HiPC 2012, Pune, India, December 18-22, 2012*, pages 1–10.

[Backer, 2007] Backer, D. (2007). Power Quality and Asset Management The Other "Two-Thirds" of AMI Value. In *Rural Electric Power Conference, 2007 IEEE*, pages C6–C6–8.

[Barrachina et al., 2013] Barrachina, S., Barreda, M., CatalÃ¡n, S., Dolz, M. F., Fabregat, G., Mayo, R., Quintana-Ortí, E. S., and Reyes, R. (2013). An integrated framework for power-performance analysis of parallel scientific workloads. In *ENERGY 2013, The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies, Lisbon, Portugal, March 24-29, 2013*, pages 114–119.

[Barroso and Hölzle, 2007] Barroso, L. A. and Hölzle, U. (2007). The case for energy-proportional computing. *IEEE Computer*, 40(12) :33–37.

[Bedard et al., 2010] Bedard, D., Lim, M. Y., Fowler, R., and Porterfield, A. (2010). PowerMon : Fine-grained and integrated power monitoring for commodity computer systems. In *Proceedings Southeastcon 2010*, Charlotte, NC. IEEE.

[Bergman et al., 2008] Bergman, K., Borkar, S., Campbell, D., and others (2008). ExaScale Computing Study : Technology Challenges in Achieving Exascale Systems. In *DARPA Information Processing Techniques Office*, page pp. 278, Washington, DC.

[Black, 1969] Black, J. (1969). Electromigration - a brief survey and some recent results. *IEEE Transactions on Electron Devices*, 16(4) :338 – 347.

[Bougeret et al., 2011] Bougeret, M., Casanova, H., Rabie, M., Robert, Y., and Vivien, F. (2011). Checkpointing strategies for parallel jobs. Research Report 7520, INRIA, France.

[Bouguerra et al., 2010] Bouguerra, M. S., Trystram, D., and Wagner, F. (2010). An optimal algorithm for scheduling checkpoints with variable costs. Technical report, INRIA.

[Bourdon et al., 2013] Bourdon, A., Noureddine, A., Rouvoy, R., and Seinturier, L. (2013). Powerapi : A software library to monitor the energy consumed at the process-level. *ERCIM News*, 2013(92).

[Bouteiller et al., 2010] Bouteiller, A., Bosilca, G., and Dongarra, J. (2010). Redesigning the message logging model for high performance. *Concurrency and Computation : Practice and Experience*, 22(16) :2196–2211.

[Bouteiller et al., 2011] Bouteiller, A., Hérault, T., Bosilca, G., and Dongarra, J. J. (2011). Correlated set coordination in fault tolerant message logging protocols. In *Euro-Par 2011 Parallel Processing - 17th International Conference, Proceedings, Part II*, volume 6853 of *Lecture Notes in Computer Science*, pages 51–64.

[Bouteiller et al., 2006] Bouteiller, A., Herault, T., Krawezik, G., Lemarinier, P., and Cappello, F. (2006). MPICH-V : a multiprotocol fault tolerant MPI. *International Journal of High Performance Computing and Applications*, 20(3) :319–333.

[Brooks et al., 2000] Brooks, D., Tiwari, V., and Martonosi, M. (2000). Wattch : a framework for architectural-level power analysis and optimizations. In *27th International Symposium on Computer Architecture (ISCA 2000), June 10-14, 2000, Vancouver, BC, Canada*, pages 83–94.

[Calderaro et al., 2011] Calderaro, V., Hadjicostis, C. N., Piccolo, A., and Siano, P. (2011). Failure identification in smart grids based on petri net modeling. *IEEE Transactions on Industrial Electronics*, 58(10) :4613–4623.

[Cappello et al., 2005] Cappello, F., Caron, E., Daydé, M. J., Desprez, F., Jégou, Y., Primet, P. V.-B., Jeannot, E., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Quétier, B., and Richard., O. (2005). Grid'5000 : A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *IEEE/ACM Grid 2005, Seattle, Washington, USA*.

[Cappello et al., 2009] Cappello, F., Geist, A., Gropp, B., Kale, S., Kramer, B., and Snir, M. (2009). Toward exascale resilience. *International Journal of High Performance Computing Applications*, 23 :374–388.

[Carpentier et al., 2012] Carpentier, J., Gelas, J.-P., Lefèvre, L., Morel, M., Mornard, O., and Laisné, J.-P. (2012). Compatibleone : Designing an energy efficient open source cloud broker. In *2012 Second International Conference on Cloud and Green Computing, CGC 2012, Xiangtan, Hunan, China, November 1-3, 2012*, pages 199–205.

[Castillo et al., 2011] Castillo, M., Dolz, M. F., Fernández, J. C., Mayo, R., Quintana-Ortí, E. S., and Roca, V. (2011). Evaluation of the energy performance

of dense linear algebra kernels on multi-core and many-core processors. In *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May 2011 - Workshop Proceedings*, pages 846–853.

[Chandy and Lamport, 1985] Chandy, K. M. and Lamport, L. (1985). Distributed snapshots : Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1) :63–75.

[Chase et al., 2001] Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., and Doyle, R. P. (2001). Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP'01, pages 103–116, Banff, Alberta, Canada.

[Chertkov et al., 2011] Chertkov, M., Pan, F., and Stepanov, M. G. (2011). Predicting failures in power grids : The case of static overloads. *IEEE Trans. Smart Grid*, 2(1) :162–172.

[Cleveland, 2008] Cleveland, F. (2008). Cyber security issues for advanced metering infrasttructure (ami). In *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, pages 1–5.

[Da-Costa et al., 2010] Da-Costa, G., Dias de Assuncao, M., Gelas, J.-P., Georgiou, Y., Lefèvre, L., Orgerie, A.-C., Pierson, J.-M., Richard, O., and Sayah, A. (2010). Multi-facet approach to reduce energy consumption in clouds and grids : The green-net framework. In *e-Energy 2010 : First International Conference on Energy-Efficient Computing and Networking*, Passau, Germany.

[Da Costa and Hlavacs, 2010] Da Costa, G. and Hlavacs, H. (2010). Methodology of Measurement for Energy Consumption of Applications. In *Energy Efficient Grids, Clouds and Clusters Workshop (co-located with Grid), E2GC2 2010, October, 25 - 29, 2010, Brussels, Belgium.*

[Daly, 2006] Daly, J. T. (2006). A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Comp. Syst.*, 22(3) :303–312.

[Department of Energy, 2007] Department of Energy, O. o. S. (2007). Modeling and simulation at the exascale for energy and the environment. Washington, D.C.

[Dias de Assuncao et al., 2010a] Dias de Assuncao, M., Gelas, J.-P., Lefèvre, L., and Orgerie, A.-C. (2010a). The green grid5000 : Instrumenting a grid with energy sensors. In *5th International Workshop on Distributed Cooperative Laboratories : Instrumenting the Grid (INGRID 2010)*, Poznan, Poland.

[Dias de Assuncao et al., 2010b] Dias de Assuncao, M., Orgerie, A.-C., and Lefèvre, L. (2010b). An analysis of power consumption logs from a monitored grid site. In *IEEE/ACM International Conference on Green Computing and Communications (GreenCom-2010)*, pages 61–68, Hangzhou, China.

[Diouri et al., 2013a] Diouri, M. E. M., Dolz, M. F., Glück, O., Lefèvre, L., Alonso, P., Catalán, S., Mayo, R., , and Quintana-Ortí, E. S. (2013a). Solving some

mysteries in power monitoring of servers : Take care of your wattmeters ! In *Energy Efficiency in Large Scale Distributed Systems (EE-LSDS), Vienna, Austria, April, 22-24 2013*.

[Diouri et al., 2011] Diouri, M. E. M., Glück, O., and Lefèvre, L. (2011). Vers des machines exaflopiques vertes. In *Actes des 20mes Rencontres francophones du Parallélisme (RenPar'20)*, Saint-Malo, France.

[Diouri et al., 2012a] Diouri, M. E. M., Glück, O., and Lefèvre, L. (2012a). Towards a novel smart and energy-aware service-oriented manager for extreme-scale applications. In 1^{st} *Workshop on Power-Friendly Grid Computing, co-located with the 2012 International Green Computing Conference, IGCC 2012, San Jose, CA, USA, June 4-8, 2012*, pages 1–6.

[Diouri et al., 2013b] Diouri, M. E. M., Glück, O., and Lefèvre, L. (2013b). Sesames : a smart-grid based framework for consuming less and better in extreme-scale infrastructures. In *The 2013 IEEE International Conference on Green Computing and Communications, Beijing, China, August 20-23, 2013*.

[Diouri et al., 2013c] Diouri, M. E. M., Glück, O., and Lefèvre, L. (2013c). Smart energy management for greener supercomputing. *ERCIM News (European Research Consortium for Informatics and Mathematics)*, 2013(92).

[Diouri et al., 2013d] Diouri, M. E. M., Glück, O., and Lefèvre, L. (2013d). Your Cluster is not Power Homogeneous : Take Care when Designing Green Schedulers ! In 4^{th} *IEEE International Green Computing Conference (IGCC), Arlington, VA USA*.

[Diouri et al., 2012b] Diouri, M. E. M., Glück, O., Lefèvre, L., and Cappello, F. (2012b). Energy considerations in checkpointing and fault tolerance protocols. In 2^{nd} *Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2012), co-located with the* 42^{th} *conference on Dependable Systems and Networks.*, Boston, USA.

[Diouri et al., 2013e] Diouri, M. E. M., Glück, O., Lefèvre, L., and Cappello, F. (2013e). ECOFIT : A Framework to Estimate Energy Consumption of Fault Tolerance protocols during HPC executions. In *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Delft, Netherlands.

[Diouri et al., 2013f] Diouri, M. E. M., Glück, O., Lefèvre, L., and Cappello, F. (2013f). Towards an Energy Estimator of Fault Tolerance protocols (poster paper). In 18^{th} *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2013)*, Shenzhen, China.

[Diouri et al., 2013g] Diouri, M. E. M., Glück, O., Lefèvre, L., and Mignot, J.-C. (2013g). Energy Estimation for MPI Broadcasting Algorithms in Large Scale HPC Systems. In 20^{th} *European MPI Users' Group Meeting on Recent Advances in Message Passing Interface (EuroMPI 2013), Madrid, Spain.*

[Diouri et al., 2013h] Diouri, M. E. M., Tsafack Chetsa, G. L., Glück, O., Lefèvre, L., Pierson, J.-M., Stolf, P., and Da Costa, G. (2013h). Energy efficiency in high-performance computing with and without knowledge of applications and services.

International Journal of High Performance Computing Applications (IJHPCA). To appear in 2013.

[Do et al., 2009] Do, T., Rawshdeh, S., and Shi, W. (2009). pTop : A process-level power profiling tool. In *HotPower '09 : Proceedings of the Workshop on Power Aware Computing and Systems*, New York, NY, USA.

[Dongarra, 1987] Dongarra, J. (1987). The LINPACK Benchmark : An Explanation. In *ICS : Proceedings of 1st International Conference Supercomputing, Athens, Greece, June 8-12, 1987*, volume 297 of *Lecture Notes in Computer Science*.

[Etinski et al., 2010] Etinski, M., Corbalan, J., Labarta, J., and Valero, M. (2010). Utilization driven power-aware parallel job scheduling. *Computer Science - Research and Development*, 25(3-4) :207–216.

[European Commission, 2006] European Commission (2006). Vision and Strategy for Europe's Electricity Networks of the Future.

[Fan et al., 2007a] Fan, X., Weber, W.-D., and Barroso, L. A. (2007a). Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 13–23, New York, NY, USA.

[Fan et al., 2007b] Fan, X., Weber, W.-D., and Barroso, L. A. (2007b). Power provisioning for a warehouse-sized computer. In *34th International Symposium on Computer Architecture (ISCA 2007), June 9-13, 2007, San Diego, California, USA*, pages 13–23.

[Feng et al., 2010] Feng, W.-c., Cameron, K. W., Scogland, T., Lin, H., and Subramaniam, B. (2010). The green500 list.

[Freeh et al., 2007] Freeh, V. W., Lowenthal, D. K., Pan, F., Kappiah, N., Springer, R., Rountree, B., and Femal, M. E. (2007). Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Trans. Parallel Distrib. Syst.*, 18(6) :835–848.

[Fu et al., 2012] Fu, Q., Montoya, L. F., Solanki, A., Nasiri, A., Bhavaraju, V., Abdallah, T., and Yu, D. C. (2012). Microgrid generation capacity design with renewables and energy storage addressing power quality and surety. *IEEE Trans. Smart Grid*, 3(4) :2019–2027.

[Ge et al., 2009] Ge, R., Feng, X., Song, S., Chang, H.-C., Li, D., and Cameron, K. W. (2009). Powerpack : Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 99 :658–671.

[Giri and Vanchi, 2010] Giri, R. and Vanchi, A. (2010). Increasing data center efficiency with server power measurements. Technical report, Intel Information Technology.

[Goiri et al., 2011] Goiri, I., Beauchea, R., Le, K., Nguyen, T. D., Haque, M. E., Guitart, J., Torres, J., and Bianchini, R. (2011). Greenslot : scheduling energy

consumption in green datacenters. In *Conference on High Performance Computing Networking, Storage and Analysis, SC 2011, Seattle, WA, USA, November 12-18, 2011*, page 20.

[Guermouche et al., 2011] Guermouche, A., Ropars, T., Brunet, E., Snir, M., and Cappello, F. (2011). Uncoordinated checkpointing without domino effect for send-deterministic mpi applications. In *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May, 2011*, pages 989 –1000.

[Guermouche et al., pear] Guermouche, A., Ropars, T., Snir, M., and Cappello, F. (to appear). HydEE : Failure Containment without Event Logging for Large Scale Send-Deterministic MPI Applications. In *Proceedings of IEEE IPDPS 2012*.

[Gunaratne and Christensen, 2006] Gunaratne, C. and Christensen, K. J. (2006). Ethernet adaptive link rate : System design and performance evaluation. In *31st IEEE Conference on Local Computer Networks, Tampa, Florida, USA, 14-16 November 2006*.

[Gungor et al., 2013] Gungor, V. C., Sahin, D., Kocak, T., Ergüt, S., Buccella, C., Cecati, C., and Hancke, G. P. (2013). A survey on smart grid potential applications and communication requirements. *IEEE Trans. Industrial Informatics*, 9(1) :28–42.

[Gurumurthi et al., 2002] Gurumurthi, S., Sivasubramaniam, A., Irwin, M. J., Vijaykrishnan, N., Kandemir, M. T., Li, T., and John, L. K. (2002). Using complete machine simulation for software power estimation : The softwatt approach. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA'02), Boston, Massachusettes, USA, February 2-6, 2002*, pages 141–150.

[Gurumurthi et al., 2003] Gurumurthi, S., Sivasubramaniam, A., Kandemir, M., and Franke, H. (2003). Drpm : dynamic speed control for power management in server class disks. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 169–179.

[Hack and Bierly, 2007] Hack, J. and Bierly, E. (2007). Computational and informational technology rate limiters to the advancement of climate change science. DOE Advanced Scientific Computing Research Advisory Committee.

[Hähnel et al., 2012] Hähnel, M., Döbel, B., Völp, M., and Härtig, H. (2012). Measuring energy consumption for short code paths using rapl. *SIGMETRICS Performance Evaluation Review*, 40(3) :13–17.

[Hermenier et al., 2006] Hermenier, F., Loriant, N., and Menaud, J.-M. (2006). Power Management in Grid Computing with Xen. In *Frontiers of High Performance Computing and Networking - ISPA 2006 Workshops, ISPA 2006 International Workshops, FHPCN, XHPC, S-GRACE, GridGIS, HPC-GTP, PDCE, ParDMCom, WOMP, ISDF, and UPW*, volume 4331 of *Lecture Notes in Computer Science*, pages 407–416, Sorrento, Italy.

[Hikita et al., 2008] Hikita, J., Hirano, A., and Nakashima, H. (2008). Saving 200kw and 200 k/year by power-aware job/machine scheduling. In *22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida USA, April 14-18, 2008*, pages 1–8.

[Hlavacs et al., 2009] Hlavacs, H., Da Costa, G., and Pierson, J.-M. (2009). Energy consumption of residential and professional switches. In *IEEE CSE*.

[Ho et al., 2008] Ho, J., Wang, C., and Lau, F. (2008). Scalable group-based checkpoint/restart for large-scale message-passing systems. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12. IEEE.

[Hotta et al., 2006] Hotta, Y., Sato, M., Kimura, H., Matsuoka, S., Boku, T., and Takahashi, D. (2006). Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. In *Proceedings of the 20th International in Parallel and Distributed Processing Symposium, IPDPS 2006*.

[Hsu and chun Feng, 2005] Hsu, C.-H. and chun Feng, W. (2005). A power-aware run-time system for high-performance computing. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*.

[Intel, 2009] Intel (2009). Intel Makes Multi-Million Euro Investment to Create European Exascale Computing Research Center, http://www.intel.com/pressroom/archive/releases/2009/20091118comp.htm.

[Intel et al., 2009] Intel, Hewlett-Packard, NEC, and Dell (2009). Ipmi specification, v2.0/1.5, rev. 4 : Addendum.

[Isci and Martonosi, 2003] Isci, C. and Martonosi, M. (2003). Runtime power monitoring in high-end processors : Methodology and empirical data. In *Proceedings of the 36th Annual International Symposium on Microarchitecture, San Diego, CA, USA, December 3-5, 2003*, pages 93–104.

[Janzen, 2001] Janzen, J. (2001). Calculating Memory System Power for DDR SDRAM. Technical note, tn-46-03, Micro Designline.

[Joseph et al., 2001] Joseph, R., Brooks, D., and Martonosi, M. (2001). Live, Runtime Power Measurements as a Foundation for Evaluating Power/Performance Tradeoffs. In *Workshop on Complexity Effective Design WCED, held in conjunction with ISCA-28*.

[Joseph and Martonosi, 2001] Joseph, R. and Martonosi, M. (2001). Run-time power estimation in high performance microprocessors. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design, 2001, Huntington Beach, California, USA, 2001*, pages 135–140.

[Kamil et al., 2008] Kamil, S., Shalf, J., and Strohmaier, E. (2008). Power efficiency in high performance computing. In *22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida USA, April 14-18, 2008*, pages 1–8.

[Kim et al., 2007] Kim, K. H., Buyya, R., and Kim, J. (2007). Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007), 14-17 May 2007, Rio de Janeiro, Brazil*, pages 541–548.

[Kim and Poor, 2011] Kim, T. T. and Poor, H. V. (2011). Scheduling power consumption with price uncertainty. *IEEE Trans. Smart Grid*, 2(3) :519–527.

[Ko and Hahn, 2013] Ko, W. and Hahn, T. (2013). Analysis of consumer preferences for electric vehicles. *IEEE Trans. Smart Grid*, 4(1) :437–442.

[Kothe, 2007] Kothe, D. B. (2007). Science Prospects and Benefits with Exascale Computing. Technical Report ORNL/TM-2007/232, OAK Ridge National Laboratory.

[Lim et al., 2010] Lim, M. Y., Porterfield, A., and Fowler, R. (2010). SoftPower : Fine-Grain Power Estimations Using Performance Counters. In *The ACM International Symposium on High Performance Distributed Computing (HPDC)*, Chicago.

[Lisovich et al., 2010] Lisovich, M. A., Mulligan, D. K., and Wicker, S. B. (2010). Inferring personal information from demand-response systems. *IEEE Security & Privacy*, 8(1) :11–20.

[Liu et al., 2012] Liu, X., Ivanescu, L., Kang, R., and Maier, M. (2012). Real-time household load priority scheduling algorithm based on prediction of renewable source availability. *IEEE Trans. Consumer Electronics*, 58(2) :318–326.

[Mahadevan et al., 2009] Mahadevan, P., Sharma, P., Banerjee, S., and Ranganathan, P. (2009). A power benchmarking framework for network devices. In *NETWORKING 2009 Conference, Aachen, Germany, May 11-15, 2009.*, pages 795–808.

[McCullough et al., 2011] McCullough, J. C., Agarwal, Y., Chandrashekar, J., Kuppuswamy, S., Snoeren, A. C., and Gupta, R. K. (2011). Evaluating the effectiveness of model-based power characterization. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 12–12.

[Meneses et al., 2010] Meneses, E., Mendes, C. L., and Kalé, L. V. (2010). Team-based message logging : Preliminary results. In *3rd Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids (CCGRID 2010).*

[Meneses et al., 2012] Meneses, E., Sarood, O., and Kalé, L. V. (2012). Assessing energy efficiency of fault tolerance protocols for hpc systems. In *SBAC-PAD 2012, New York, NY, USA, October 24-26, 2012*, pages 35–42.

[Meuer, 2000] Meuer, H. W. (2000). The TOP500 Project of the Universities Mannheim and Tennessee. In *Euro-Par 2000, Proceedings of the 6th International Euro-Par Conference on Parallel Processing, Munich, Germany, August 29 - September 1, 2000*, number 1900 in Lecture Notes in Computer Science, pages 43–33.

[Molaro et al., 2009] Molaro, D., Payer, H., and Le Moal, D. (2009). Tempo : Disk drive power consumption characterization and modeling. In *Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on*, pages 246–250.

[Moody et al., 2010] Moody, A., Bronevetsky, G., Mohror, K., and Supinski, B. R. d. (2010). Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *Proceedings of the ACM/IEEE SC Conference*, pages 1–11.

[Navarro et al., 2011] Navarro, J., Armendáriz-Iñigo, J. E., and Climent, A. (2011). Dynamic distributed storage architecture on smart grids. In *ICSOFT 2011 - Proceedings of the 6th International Conference on Software and Data Technologies, Volume 1, Seville, Spain, 18-21 July, 2011*, pages 218–221.

[Ndiaye et al., 2012] Ndiaye, N. M., Sens, P., and Thiare, O. (2012). Performance comparison of hierarchical checkpoint protocols grid computing. In *Distributed Computing and Artificial Intelligence - 9th International Conference, DCAI 2012, Salamanca, Spain, 28-30th March, 2012*, pages 339–346.

[Netzer and Xu, 1995] Netzer, R. H. B. and Xu, J. (1995). Necessary and sufficient conditions for consistent global snapshots. *IEEE Transactions on Parallel and Distributed Systems*, 6(2) :165–169.

[Orgerie and Lefèvre, 2009] Orgerie, A.-C. and Lefèvre, L. (2009). When clouds become green : the green open cloud architecture. In *Parco2009 : International Conference on Parallel Computing*, Lyon, France.

[Orgerie and Lefèvre, 2011] Orgerie, A.-C. and Lefèvre, L. (2011). Eridis : Energy-efficient reservation infrastructure for large-scale distributed systems. *Parallel Processing Letters*, 21(2) :133–154.

[Orgerie et al., 2008a] Orgerie, A.-C., Lefèvre, L., and Gelas, J.-P. (2008a). Chasing gaps between bursts : Towards energy efficient large scale experimental grids. In *Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2008, Dunedin, Otago, New Zealand, 1-4 December 2008*, pages 381–389.

[Orgerie et al., 2008b] Orgerie, A.-C., Lefevre, L., and Gelas, J.-P. (2008b). Save Watts in your Grid : Green Strategies for Energy-Aware Framework in Large Scale Distributed Systems. In *ICPADS 2008 : The 14th IEEE International Conference on Parallel and Distributed Systems, Melbourne, Australia.*

[Ozturk et al., 2013] Ozturk, Y., Senthilkumar, D., Kumar, S., and Lee, G. K. (2013). An intelligent home energy management system to improve demand response. *IEEE Trans. Smart Grid*, 4(2) :694–701.

[Pinheiro et al., 2001] Pinheiro, E., Bianchini, R., Carrera, E. V., and Heath, T. (2001). Load balancing and unbalancing for power and performance in cluster-based systems. In *IN WORKSHOP ON COMPILERS AND OPERATING SYSTEMS FOR LOW POWER.*

[Pjesivac-Grbovic et al., 2005] Pjesivac-Grbovic, J., Angskun, T., Bosilca, G., Fagg, G. E., Gabriel, E., and Dongarra, J. (2005). Performance Analysis of MPI Collective Operations. In *IEEE IPDPS 2005, 4-8 April 2005, Denver, CO, USA*.

[Rabenseifner et al., 2009] Rabenseifner, R., Hager, G., and Jost, G. (2009). Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 427 –436.

[Rao et al., 1999] Rao, C., Toutenburg, H., Fieger, A., Heumann, C., Nittner, T., and Scheid, S. (1999). Linear models : Least squares and alternatives. *Springer Series in Statistics*.

[Ropars et al., 2011] Ropars, T., Guermouche, A., Uçar, B., Meneses, E., Kalé, L. V., and Cappello, F. (2011). On the use of cluster-based partial message logging to improve fault tolerance for mpi hpc applications. In *17th International Conference on Parallel Processing, Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011*, pages 567–578.

[Rotem et al., 2012] Rotem, E., Naveh, A., Ananthakrishnan, A., Weissmann, E., and Rajwan, D. (2012). Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2) :20–27.

[Saber and Venayagamoorthy, 2012] Saber, A. Y. and Venayagamoorthy, G. K. (2012). Resource scheduling under uncertainty in a smart grid with renewables and plug-in vehicles. *IEEE Systems Journal*, 6(1) :103–109.

[Sharma et al., 2006] Sharma, S., Hsu, C.-H., and Feng, W.-c. (2006). Making a case for a green500 list. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp.

[Sloot et al., 2003] Sloot, P. M. A., Abramson, D., Bogdanov, A. V., Dongarra, J., Zomaya, A. Y., and Gorbachev, Y. E., editors (2003). *Computational Science - ICCS 2003, International Conference, Melbourne, Australia and St. Petersburg, Russia, June 2-4, 2003. Proceedings, Part IV*, volume 2660 of *Lecture Notes in Computer Science*.

[Steinder et al., 2008] Steinder, M., Whalley, I., Hanson, J. E., and Kephart, J. O. (2008). Coordinated management of power usage and runtime performance. In *IEEE/IFIP Network Operations and Management Symposium : Pervasive Management for Ubioquitous Networks and Services, NOMS 2008, 7-11 April 2008, Salvador, Bahia, Brazil*, pages 387–394.

[Tang et al., 2007] Tang, Q., Gupta, S. K. S., and Varsamopoulos, G. (2007). Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *Proceedings of the 2007 IEEE International Conference on Cluster Computing, 17-20 September 2007, Austin, Texas, USA (CLUSTER 2007)*, pages 129–138.

[Thakur and Gropp, 2003] Thakur, R. and Gropp, W. (2003). Improving the Performance of Collective Operations in MPICH. In *European PVM/MPI Users' Group Meeting, Venice, Italy, September 29 - October 2, 2003*, pages 257–267.

[Thimmapuram and Kim, 2013] Thimmapuram, P. R. and Kim, J. (2013). Consumers' price elasticity of demand modeling with economic effects on electricity markets using an agent-based model. *IEEE Trans. Smart Grid*, 4(1) :390–397.

[Tsui and Chan, 2012] Tsui, K. M. and Chan, S. C. (2012). Demand response optimization for smart home scheduling under real-time pricing. *IEEE Trans. Smart Grid*, 3(4) :1812–1821.

[Varsamopoulos et al., 2009] Varsamopoulos, G., Banerjee, A., and Gupta, S. K. S. (2009). Energy efficiency of thermal-aware job scheduling algorithms under various cooling models. In *Second International Conference on Contemporary Computing, IC3 2009, Noida, India, August 17-19, 2009*, volume 40 of *Communications in Computer and Information Science*, pages 568–580.

[Viredaz and Wallach, 2000] Viredaz, M. and Wallach, D. (2000). Power evaluation of itsy version 2.3. Technical report, Compaq, Western Research Laboratory.

[Wadsworth and Chen, 2008] Wadsworth, D. M. and Chen, Z. (2008). Performance of MPI broadcast algorithms. In *IEEE IPDPS 2008, Miami, Florida USA, April 14-18, 2008*, pages 1–7.

[Wang et al., 2002] Wang, H., Zhu, X., Peh, L.-S., and Malik, S. (2002). Orion : a power-performance simulator for interconnection networks. In *Proceedings of the 35th Annual International Symposium on Microarchitecture, Istanbul, Turkey, November 18-22, 2002*, pages 294–305.

[Xu et al., 2012] Xu, Z., Guan, X., Jia, Q.-S., Wu, J., Wang, D., and Chen, S. (2012). Performance analysis and comparison on energy storage devices for smart building energy management. *IEEE Trans. Smart Grid*, 3(4) :2136–2147.

[Ye et al., 2002] Ye, T. T., Micheli, G. D., and Benini, L. (2002). Analysis of power consumption on switch fabrics in network routers. In *Proceedings of the 39th Design Automation Conference, DAC 2002, New Orleans, LA, USA, June 10-14, 2002*, pages 524–529.

[Ye et al., 2000] Ye, W., Vijaykrishnan, N., Kandemir, M. T., and Irwin, M. J. (2000). The design and use of simplepower : a cycle-accurate energy estimation tool. In *DAC*, pages 340–345.

[Young, 1974] Young, J. W. (1974). A first order approximation to the optimum checkpoint interval. *Commun. ACM*, 17(9) :530–531.

[Yu et al., 2012] Yu, R., Yang, W., and Rahardja, S. (2012). A statistical demand-price model with its application in optimal real-time price. *IEEE Trans. Smart Grid*, 3(4) :1734–1742.

[Zedlewski et al., 2003] Zedlewski, J., Sobti, S., Garg, N., Zheng, F., Krishnamurthy, A., and Wang, R. Y. (2003). Modeling hard-disk power consumption. In *Proceedings of the FAST '03 Conference on File and Storage Technologies, March 31 - April 2, 2003, Cathedral Hill Hotel, San Francisco, California, USA*.

[Zeng et al., 2012] Zeng, L., Veeravalli, B., and Li, X. (2012). Scalestar : Budget conscious scheduling precedence-constrained many-task workflow applications in

cloud. In *IEEE 26th International Conference on Advanced Information Networking and Applications, AINA, 2012 , Fukuoka, Japan, March 26-29, 2012*, pages 534–541.

Résumé

Les infrastructures de calcul très haute performance ont connu une croissance rapide en particulier ces dernières années. Cette croissance a toujours été motivée par les besoins accrus en puissance de calcul qu'expriment les scientifiques dans divers domaines. Cependant, ces systèmes devenus de plus en plus larges constituent de gros consommateurs d'électricité et consomment déjà plusieurs mégawatts. Afin de consommer "moins" et "mieux", nous avons proposé un environnement logiciel qui d'une part, permet de choisir avant de pré-exécuter l'application, les versions de services applicatifs consommant le moins d'énergie, et qui d'autre part, repose sur une grille électrique intelligente pour planifier les réservations des ressources de calcul de ces infrastructures. Cet environnement, appelé SESAMES, a été adapté à deux services applicatifs indispensables au calcul très haute performance : la tolérance aux pannes et la diffusion de données. Des validations expérimentales ont montré que l'on peut réduire la consommation énergétique de chacun des deux services étudiés en s'appuyant sur les estimations énergétiques précises fournies par SESAMES pour n'importe quel contexte d'exécution et pour n'importe quelle plate-forme dotée de wattmètres. Notre méthodologie d'estimation repose sur une description du contexte d'exécution et sur une calibration de la plate-forme d'exécution basée sur la collecte de mesures énergétiques. Des simulations ont démontré que l'ordonnanceur multi-critères des réservations de ressources proposé dans SESAMES, permet de réduire à la fois la consommation énergétique, le coût financier et l'impact environnemental de ces réservations, tout en respectant les contraintes imposées par l'utilisateur et le fournisseur d'énergie.

Mots-clés : évaluation énergétique ; efficacité énergétique ; calcul intensif ; tolérance aux pannes ; diffusion de données ; ordonnancement.

Abstract

High-Performance Computing (HPC) infrastructures have experienced a rapid growth, particularly these last years. This growth has been driven by the increased need in terms of computational power expressed by scientists in various fields. However, their increasing size makes them important electricity consumers since they already consume several megawatts. In order to consume "less" and better", we proposed a framework which permits to choose the less energy consuming versions of the services before pre-executing the application. In addition, our framework relies on a smart grid in order to schedule resource reservations on these computing infrastructures. This framework, called SESAMES, is adapted to two services required in high performance computing : fault tolerance and data broadcasting. Experimental validations have shown that we can reduce the energy consumption of both services by relying on accurate energy estimations provided SESAMES for any execution context and for any platform equipped with wattmeters. Our estimation methodology is based on a description of the execution context and on a platform calibration that consists of gathering energy measurements. Simulations have shown that the multi-criteria reservation scheduler proposed in SESAMES, simultaneously reduces the energy consumption, the financial cost and the environmental impact of these reservations, while respecting the constraints imposed by the user and the energy supplier.

Keywords : energy evaluation ; energy efficiency ; high performance computing ; fault tolerance ; data broadcasting ; scheduling.