Communauté UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Millian POQUET

Thèse dirigée par **Denis TRYSTRAM**
et codirigée par **Pierre-François DUTOT**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
et de l'École Doctorale **MSTII**

Approche par la simulation pour la gestion de ressources

Simulation approach for resource management

Thèse soutenue publiquement le **19 décembre 2017**,
devant le jury composé de :

Henri CASANOVA
Professor, University of Hawai'i at Mānoa, États-Unis, Rapporteur
Georges DA COSTA
Maître de conférences, IRIT, Univ. Toulouse III, France, Rapporteur
Yves DENNEULIN
Professeur des universités, LIG, Grenoble INP, France, Examinateur
Frédéric DESPREZ
Directeur de recherche, LIG, Inria, France, Président
Pierre-François DUTOT
Maître de conférences, LIG, Univ. Grenoble Alpes, France, Co-Directeur de thèse
Sascha HUNOLD
Assistant professor, Technische Universität Wien, Autriche, Examinateur
Anne-Cécile ORGERIE
Chargée de recherche, IRISA, CNRS, France, Examinatrice
Olivier RICHARD
Maître de conférences, LIG, Univ. Grenoble Alpes, France, Invité
Denis TRYSTRAM
Professeur des universités, LIG, Grenoble INP, France, Directeur de thèse

> *The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom.*
>
> — **Isaac Asimov**

Remerciements

(Acknowledgments)

I would like to thank all the members of the jury for their helpful feedback and comments — and in particular Henri CASANOVA and Georges DA COSTA for reviewing the whole dissertation.

Un grand merci tout d'abord à Denis et Pierre-François, qui ont eu la patience de me supporter tout au long de cette thèse — il en fallait. Leur bienveillance et leurs conseils éclairés et constants sur de nombreux domaines m'ont beaucoup appris et ont vivement amélioré la qualité de ce travail. Je tiens à remercier tous les membres des équipes DATAMOVE/POLARIS (débuté en MOAÏS/MESCAL pour moi) pour ce formidable cadre de recherche et de *troll* de toutes sortes. Merci en particulier à Olivier pour son encadrement technique — et pour ne pas m'avoir banni de son bureau malgré le temps que j'y ai passé. Merci à Arnaud pour ses conseils pertinents et pour nous trouver du temps dans cet emploi du temps surchargé. La qualité technique de ce travail serait bien amoindrie sans la bienveillance et l'abnégation de Michael, merci beaucoup. Mille mercis à Annie, pour sa sympathie et le profond soutien qu'elle nous apporte. Merci à Raphaël pour avoir contribué au présent manuscrit en se proposant comme muse entre deux tests. Merci à Fernando pour ces bons moments gastronomiques et sportifs — et félicitations pour avoir réussi à nous cacher ta maîtrise du français. Dois-je remercier Pierre pour ses nombreux éclairages techniques et esthétiques ou bien le *troller* ? Merci à Grégory pour toujours réussir à nous redonner le sourire — grâce à des blagues ou du chocolat. Merci à Valentin pour ses trouvailles logicielles et pour toujours arriver à nous surprendre avec ses histoires déroutantes. Merci à Bruno pour son aide et ses conseils. Merci à Ziad, Giorgio et David pour cet accueil chaleureux. Enfin, merci à Adrien, Christian, Clément, Danilo, Kostas, Salah, Stephan et aux Lucas pour leur sympathie.

De grands mercis à Sébastien LIMET et Sophie ROBERT pour leur sympathie et pour m'avoir fait découvrir et aimer la recherche.

Je tiens aussi à remercier ma famille pour son soutien — en particulier ma mère, ma sœur et mon père. Il en va de même pour mes amis et mes proches. Merci en particulier à Jérôme, Julien, Nicolas, Leo, Orane, Sophie et Virginie pour m'avoir supporté ces dernières années.

The work is partially supported by the ANR project MOEBUS.

Most of the experiments presented in this dissertation were were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `https://www.grid5000.fr`).

We thank the numerous contributors of the Parallel Workloads Archive and in particular Dror FEITELSON to gather these traces all together.

Abstract / Résumé

Abstract

Computing platforms increasingly grow in power and complexity. Numerous challenges remain to build next generations of platforms, but exploiting the platforms is a challenge per se. Constraints such as energy consumption, data movements and resilience risk to initiate breaking points in the way that the platforms are managed — especially with the convergence of the different types of distributed platforms.

Resource and Jobs Management Systems (RJMSs) are critical middlewares that allow users to exploit the resources of such platforms. They must evolve to make the best use of the computing platforms while complying with these new constraints. Each evolution ideally require many iterations, but conducting them *in vivo* is not reasonable due to huge overhead. Simulation is an efficient way to tackle the subsequent problems, but particular caution must be taken when drawing results from simulation as using ill-suited models may lead to invalid results.

The first contribution of this dissertation is the proposition of a modular simulation methodology to study RJMSs and their evolution realistically — and the related simulator Batsim. The main idea is to strongly separate the simulation from the decision-making algorithms. This allows separation of concerns as any algorithm can benefit from a validated simulation with multiple levels of realism (features, accuracy of the models). This methodology improves the production launch of new policies since both academic prototypes and production RJMSs can be studied in the same context.

Batsim is used in the second part of this dissertation, which focuses on online and non-clairvoyant resource management policies to save energy. Several algorithms are first proposed and analyzed to maximize performances under an energy budget for a given time period. This dissertation then explores more generally possible energy and performances trade-offs that can be obtained with node shutdown techniques.

Résumé

Les plateformes de calcul se multiplient, grandissent en taille et gagnent en complexité. De nombreux défis restent à relever pour construire les prochaines générations de plateformes, mais exploiter cesdites plateformes est également un défi en soi. Des contraintes comme la consommation énergétique, les mouvements de données ou la résilience risquent de devenir prépondérantes et de s'ajouter à la complexité actuelle de la gestion des plateformes. Les méthodes de gestion de ressources peuvent également évoluer avec la convergence des différents types de plateformes distribuées.

Les gestionnaires de ressources sont des systèmes critiques au cœur des plateformes qui permettent aux utilisateurs d'exploiter les ressources. Les faire évoluer est nécessaire pour exploiter au mieux les ressources en prenant en compte ces nouvelles contraintes. Ce processus d'évolution est risqué et nécessite de nombreuses itérations qu'il semble peu raisonnable de réaliser *in vivo* tant les coûts impliqués sont importants. La simulation, beaucoup moins coûteuse, est généralement préférée pour faire ce type d'études mais pose des questions quant au réalisme des résultats ainsi obtenus.

La première contribution de cette thèse est de proposer une méthode de simulation modulaire pour étudier les gestionnaires de ressources et leur évolution — ainsi que le simulateur résultant nommé Batsim. L'idée principale est de séparer fortement la simulation et les algorithmes de prise de décision. Cela permet une séparation des préoccupations puisque les algorithmes, quels qu'ils soient, peuvent bénéficier d'une simulation validée proposant différents niveaux de réalisme. Cette méthode simplifie la mise en production de nouvelles politiques puisque des codes issus à la fois de gestionnaires de ressources de production et de prototypes académiques peuvent être étudiés dans le même contexte.

La méthode de simulation proposée est illustrée dans la seconde partie de cette thèse, qui s'intéresse à des problèmes de gestion de ressources non clairvoyants mêlant optimisation des performances et de la consommation énergétique. Différents algorithmes sont d'abord proposés et étudiés afin de respecter un budget d'énergie pendant une période de temps donnée. Nous étudions ensuite plus généralement les différents compromis réalisables entre performances et énergie grâce à différentes politiques d'extinction de nœuds de calcul.

Contents

1 Introduction

1.1 Background

Computing platforms increasingly grow in power and complexity. **H**igh **P**erformance **C**omputing (HPC) systems continue their evolution towards bigger and more powerful platforms — future supercomputers are likely to reach *exascale* by offering a computing power of 10^{18} flop/s. Properly building and exploiting platforms of that scale is very challenging, as current designs and technologies draw near to their breaking point regarding several constraints [Don+11] — e.g., energy consumption, data movements and resilience. Simultaneously, the popularization of data centers and smaller-scale clusters broadens the computing platforms spectrum and raises new questions and challenges in terms of resource management.

The exploitation of the computing platforms is traditionally conducted within **R**esources and **J**obs **M**anagement **S**ystems (RJMSs). These complex middlewares are the very core of the platform management and play many roles, which can be grouped into two main classes. They are firstly in charge of *doing* the numerous technical procedures that occur on the system, whether they are related to the management of the users, of the jobs or of the different types of resources — e.g., processors, computing nodes, network switches or the storage system. These management procedures notably include the launching of the jobs, the monitoring of the various resources, the handling of the users' submissions and data, and the reporting of the platform current state and expected schedule. They secondly determine how the resources are shared among the different users. RJMSs are therefore the place to *take* management decisions and to implement management policies and algorithms.

RJMSs must evolve regarding both aforementioned aspects to continue to make the best use of the computing platforms. Several RJMSs design choices are not expected to scale any longer if the platforms continue to grow in size [Don+11], such as the quasi-non-distributed decision-making approach adopted by most of the RJMSs used in the TOP500 [@top500] supercomputers. Furthermore, some problems already visible on current supercomputers become increasingly important and are expected to become predominant [Luc+14]. In particular, building an exascale supercomputer

with current technology and design would lead to a tremendous power consumption — in the order of 100 MW if a power efficiency similar to the Piz Daint[1] platform's is achieved. Such immense needs are not reasonable and are likely to initiate breaking points in the way that supercomputers are both managed and designed [Ash+10]. Energy can be taken into account by the RJMSs, as it can be considered as a resource just as CPUs, memory or communication channels [Geo+15; Cha+01]. We focus on this particular aspect during this dissertation.

Making changes in RJMSs is very complex, as many other aspects must be considered. For example, scalability, heterogeneity and dynamicity keep making RJMSs ever more complicated. Another phenomenon to consider is the convergence of the different types of distributed systems that are HPC, Cloud and Big Data platforms. The underlying resource management strategies may simultaneously converge and allow more and more dedicated features to optimize the platform usage and the user quality of service. Moreover, making subsequent modifications on production RJMSs is even harder, as these critical systems should remain highly reliable. Each evolution ideally requires many real-scale test iterations, but conducting related experiments *in vivo* is most of the time impossible because of the enormous time — and energy — costs that would be involved. The most efficient manner to tackle these problems is simulation, which is faster than real experiments by multiple orders of magnitude, thus allowing the reproducible exploration of a multitude of parameters. However, particular caution must be taken when drawing conclusions from simulation [Flo06], as using ill-suited models or neglecting the models calibration may lead to invalid results.

1.2 Simulating RJMSs

A lot of studies have been devoted to the simulation of RJMSs at various stages. Most of them focus on resource management strategies and especially on scheduling and allocating jobs [KR10; Ngo+16]. Other specifically target networks [PM05; @ns3] or power management strategies [Ell+17]. Most of the existing simulators are developed by local teams for dedicated purposes and are assessed on synthetic or random data. We believe that they are too specialized to study the big picture about the evolution of RJMSs.

[1] As June 2017, the Piz Daint supercomputer is both ranked as the third most powerful platform of the TOP500 and the sixth greenest platform of the GREEN500. It develops nearly 20 petaflops per second and achieves a 10.398 Gflop/J power efficiency [@pizdaint].

Most HPC RJMSs increasingly gather features and eventually become convoluted systems with sprawling colossal amounts of source code whose modification is complex and costly. New resource management approaches and tools emphasize modularity, notably from industry [Bur+16; Hin+11; Vav+13] but also from the HPC community [Ahn+14]. Even if previous generations of RJMSs are likely to subsist for some time we believe that modular approaches will be adopted in HPC centers once mature and efficient enough — as they can drastically reduce maintenance costs. Modularity facilitates the assessment of RJMSs by simulation. Flux [Ahn+14] for example directly includes a simulator, while some RJMSs can be assessed by external tools [Liu+15]. However, implementing RJMSs and simulating the underneath computing platforms are very distinct and complex problems. We believe that coupling too strongly these problems together leads to naive simulation models that cannot reflect realistic phenomena. **The first contribution of this dissertation is to propose a modular simulation methodology to study RJMSs and their evolutions realistically — and the related simulator Batsim.**

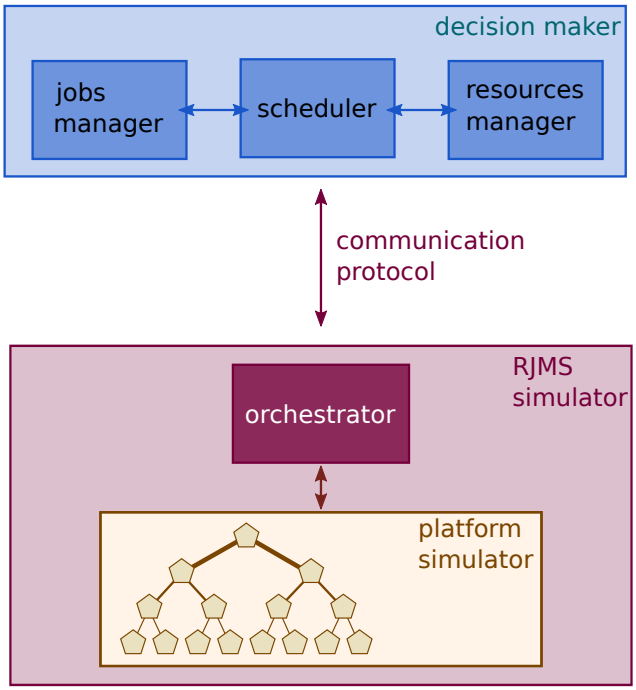


Figure 1.1: Overview of the proposed simulation methodology. The decision making is strongly decoupled from the simulation. The simulator itself can be built upon existing distributed platforms simulation frameworks.

As depicted on figure 1.1, the main idea of this methodology is to maximize separation of concerns by strongly separating the platform simulation from the decision-making procedures that decide how the platform is managed. This separation

prevents duplication of effort regarding simulation, as many decision-making procedures can be assessed with the same simulator — whether they come from real RJMSs or academic prototypes. We therefore believe that the approach can improve the use in production of new functionalities or algorithms, as academic prototypes developed with it are very likely to be compatible with real RJMSs. In fact, production RJMSs can be adapted to use the simulation side of the proposed communication protocol and therefore use any compatible decision-making algorithm. This approach is also convenient to develop new functionalities as the decoupling allows one to implement the algorithms in any programming language. The methodology additionally allows to take full advantage of existing work about the simulation of distributed systems. We strongly believe that this is the way to go to build general-purpose sound simulators, as it avoids the classical error of poorly reimplementing how complex objects should be simulated.

As said previously, we implemented the proposed simulation methodology in the Batsim simulator. Batsim is completely open source and available online[2]. It is based on the SimGrid [Cas+14] simulation framework, which allows to observe various phenomena soundly thanks to several adequate simulation models. Batsim clearly separates the way jobs are simulated from their external description, which allows multiple levels of realism with exactly the same workload descriptions — at least from the decision-making component point of view. We rely on current software engineering techniques to enhance confidence in the simulation results. In other words, substantial investment has been made in Batsim's implementation and the working state of its features is regularly and automatically tested thoroughly. Batsim is used in the second part of this dissertation, which focuses on online and non-clairvoyant resource management policies to save energy. Notice that Batsim is a general-purpose tool that can be used to conduct manifold studies involving distributed platforms and their management, and is therefore not limited to energy-related use cases. It is used to conduct several experiments in our team and outside it — e.g., in Darmstadt, Hawai'i and Lyon. As I write these lines one external article relying on Batsim for its experimental part has been published [NS17].

[2]Batsim is distributed under the LGPL-3.0 license. The project source code and documentation are available on Github [@batgit1] and on Inria's Gitlab [@batgit2]. Release versions can directly be used with the oarteam/batsim docker container.

1.3 Energy and Performances

As computing platforms grow in size, this is the same for their power consumption. Only little interest has been shown in power consumption of older generations of platforms [OLG08], notably because the involved amounts of energy remained reasonable and because the institutions in charge of such systems have the financial and technical means to manage this issue. This situation however started to change because the energy consumption of current systems becomes significant enough to be a limit to build bigger platforms. For example, the Sunway TaihuLight supercomputer — current leading system of the TOP500 — develops 93 petaflop/s and consumes 15.4 megawatts (MW) [@sunwaytl]. The second ranked system of the TOP500 is the Tianhe-2 supercomputer, which develops 33 petaflop/s and consumes 17.8 MW [@tianhe2]. Building an exascale platform with such power efficiency would lead to tremendous power consumption — a quick computation gives 165 MW from Sunway TaihuLight's power efficiency and 525 MW from Tianhe-2's. Fortunately, huge improvements on power efficiency have been conducted over the last 10 years but it is still a big challenge.

Initiated in 2007, the GREEN500 [@green500] ranks the platforms of the TOP500 by their power efficiency. As depicted in figure 1.2, the power efficiency of the TOP500 greenest platforms — the most efficient ones in terms of power efficiency — has been multiplied by 28 since 2007. These improvements affect all the types of platforms of the TOP500. For example, as June 2017, the Piz Daint supercomputer [@pizdaint] is simultaneously the third most powerful TOP500 platform and the sixth greenest one. Most recent energy efficiency improvements directly result from the hardware technology choice of including accelerators — notably **G**eneral-**P**urpose **G**raphics **P**rocessing **U**nits (GPGPUs) — as seen on figure 1.3. The recent 2.3x increase jump from June 2016 to June 2017 is for example mostly due to the adoption of Nvidia P100 GPGPUs [@greenlap2017]. Such accelerators can compute massively parallel code with a way higher energy efficiency than classical CPUs but are not adapted to all types of computations.

Reducing the power consumption of platforms can be done at several levels — namely at the hardware level, at the application level and at the platform management level. As mentioned previously, the first way to reduce energy consumption is to build platforms with more efficient hardware components. This can be done by selecting more efficient CPUs or dedicated accelerators such as GPGPUs. The other ways to reduce the energy consumption are mostly related to the resources management. This management firstly appears at the job level, as jobs must be able to make

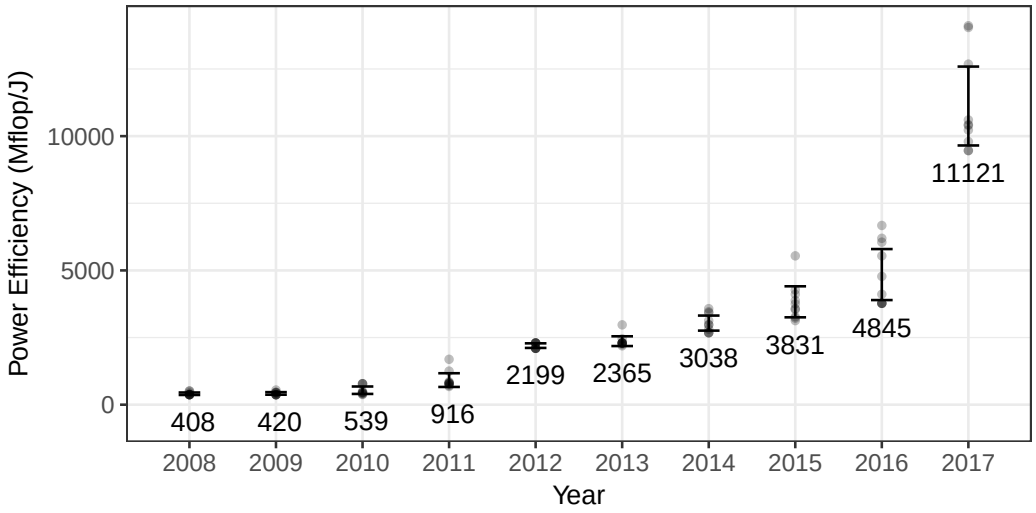


Figure 1.2: Evolution of the power efficiency of the top 10 GREEN500 platforms since 2008. The mean power efficiency is labeled in black for each year. Data has been retrieved from the GREEN500 website [@green500].

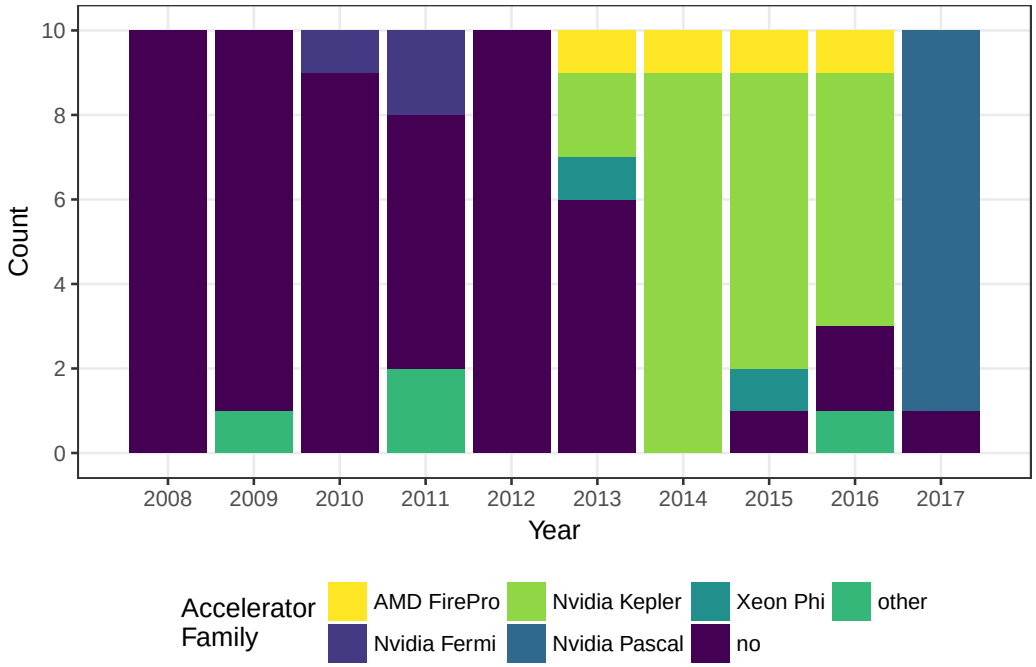


Figure 1.3: Evolution of the inclusion of accelerators in the top 10 GREEN500 platforms since 2008. Data has been retrieved from the GREEN500 website [@green500].

the most of all the resources that are allocated to them. Many job-level optimizations are possible such as reducing the job execution time or using different types of available resources when profitable. We also place the fine management of processor states — speed scaling a.k.a. **D**ynamic **V**oltage and **F**requency **S**caling (DVFS) — into the job-level optimizations category. As some applications consume more energy — integral of power over the job execution time — in lower-consumption states than in a *classical* state, we do not believe that DVFS is applicable efficiently without precise knowledge about the applications [SRH05] — information that is only available at the job level. **This dissertation addresses the problem of reducing energy at the platform management level.** At this level saving energy can be achieved by selecting where, when and how the jobs should be executed, or by taking the decision to let resources idle or to shutdown them. We think that saving energy at this level is very promising, as the possible energy savings are significant and fully compatible with hardware and job-level energy optimizations.

The first energy-related problem addressed in this dissertation consists in the maximization of the system performances under an energy budget constraint. This constraint is close to *power capping*, which consists in limiting the maximum instantaneous power consumption of the system. The difference is that here the amount of available energy is limited during a given time period rather than the instantaneous power consumption, which lets allows more room for optimizations. The resource management strategies we propose save energy by choosing not to use certain resources — by either keeping them idle or by switching them off. These strategies are extensions to the backfilling mechanism commonly implemented on HPC schedulers [MF01]. They are valid regarding the energy budget constraint while allowing better performances than a rigid power cap over the constrained time period.

This dissertation then explores more generally possible energy and performances trade-offs that can obtained with node shutdown techniques. We propose to this end different extensions to the EASY backfilling algorithm [MF01] that combine two energy saving techniques. The first technique — said opportunistic — consists in switching-off any node that remains idle for too long. The second technique explicitly chooses how many machines should remain on or off. Both techniques — and their combination — allow energy savings of the order of 20 % without significant deterioration of the performances. Our experiments show that the second technique is very promising if the number of switched-off nodes is dynamically adjusted to the system load. This technique has the major advantage of being predictable, which is not the case for the opportunistic one.

1.4 Content

The remainder of this dissertation is organized as follows. Chapter 2 first gives an overview of the different problems studied during this dissertation, presents scheduling models and objectives, and defines most of the notations used in the other chapters. Chapter 3 discusses the existing approaches to simulate RJMSs and motivates what led us to develop a new simulation methodology. The proposed simulation methodology and its associated simulator Batsim are presented in chapter 4, as well as a validation experiment that compares Batsim to the real RJMS OAR. Chapter 5 exhibits a first use of Batsim and points out that insights gained while studying theoretical models are sometimes at odd with the practical results due to shortcomings in the models. Chapter 6 includes our work about the problem of maximizing the system performances subject to an energy constraint for a given period of time. Chapter 7 analyzes the energy and performance trade-offs that can be obtained via shutdown techniques on top of the EASY backfilling algorithm. Finally, chapter 8 concludes this dissertation and gives perspectives for future work.

Each of the previous results have been published — see details in the Bibliography at the end of the manuscript.

2 Problems, Models and Notations

Work on several resource management problems is conducted in this dissertation. This chapter sketches the context of the problems and centralizes common notations and models about them.

We are interested in online resource management problems on computing platforms. Users want to use the platform to compute *work*, which is divided into *jobs* directly submitted by the users. In most distributed platforms, as shown on figure 2.1, a Resource and Jobs Management System — RJMS in short — collects data on the submitted jobs, analyzes them and finally makes decisions thanks to an online scheduling algorithm. Such scheduling algorithms are called in response to events coming from the platform — e.g., resources become available — or the users — e.g., a new job is submitted.

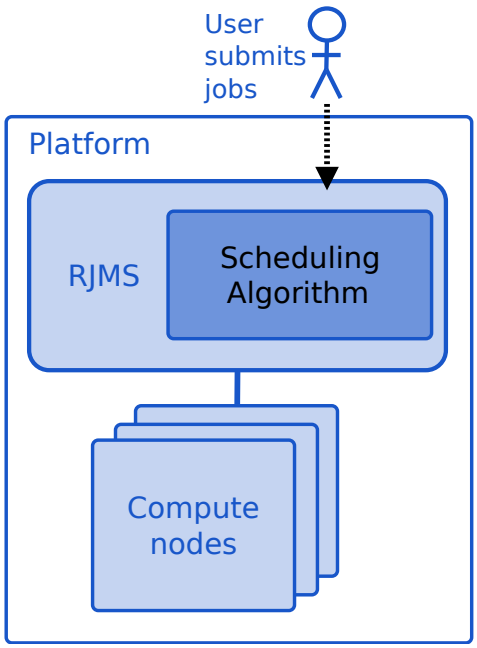


Figure 2.1: Simplified view of a HPC-like platform. Users communicate with the RJMS, which orchestrates how the resources are used.

The decisions made by the scheduling algorithm include the allocation of jobs to resources and the management of the resource states, notably by switching-off or switching-on the computation nodes. Since in the HPC context the computational resources are affected to unique jobs, only $idle$ computational resources can be selected to compute new jobs. The result of the execution of a scheduling algorithm on a suite of events can be visualized as a Gantt chart, as seen on figure 2.2. Gantt charts represent the allocation of resources to jobs over time.

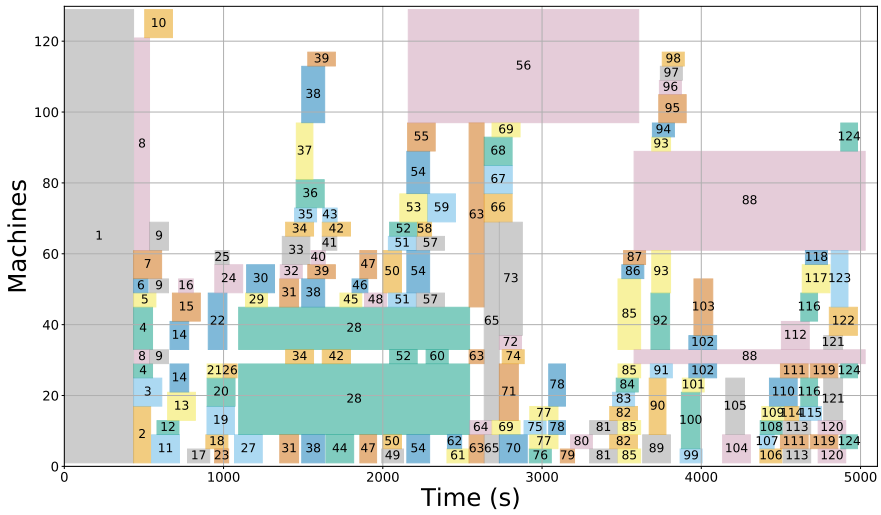


Figure 2.2: Gantt chart example. Machine utilization is plotted against time. Contiguous machine allocations are represented by rectangles. Jobs whose allocation is a set of contiguous resources — e.g., jobs 1 and 56 — are therefore represented as unique rectangles. Several rectangles sharing their time slot are used to represent non-contiguous allocations — e.g., for jobs 28, 38 and 88.

2.1 Job Characteristics

In the context of this dissertation, a job is a schedulable amount of work. Executing a job may require several computational resources. The users submit the jobs on-the-fly, stating the fixed amount of resources the jobs require. In other words, according to notations from book [Fei15], we consider parallel and rigid jobs that are submitted online.

Jobs are indexed by j in a set J. Jobs are characterized by their release time (denoted by r_j), their number of requested computing resources (q_j), and their wall-time ($wall_j$). Both q_j and $wall_j$ are specified by the users and are not known to the

scheduler before r_j. The processing time (denoted by p_j) remains unknown until the job is completed. Depending on the job computation model (cf. section 4.4), p_j can either be a fixed amount of time or depend on the execution context — depending on the selected allocation of resources and the computational and network saturation of the resources while the job is running. Job j is killed if it reaches its wall-time, without any penalty on the scheduler — in this case we simply define p_j as equal to $wall_j$. The jobs are not preemptive, which means that once started a job cannot be interrupted until its completion. Figure 2.3 outlines the main job notations.

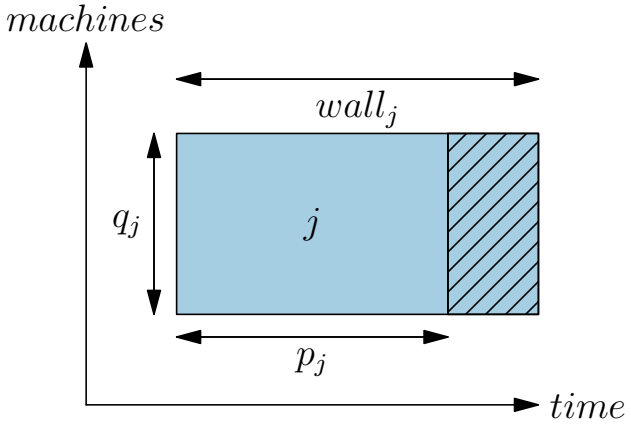


Figure 2.3: Job j requests q_j resources for a $wall_j$ amount of time.
The scheduler is unaware that j finishes after a p_j amount of time.

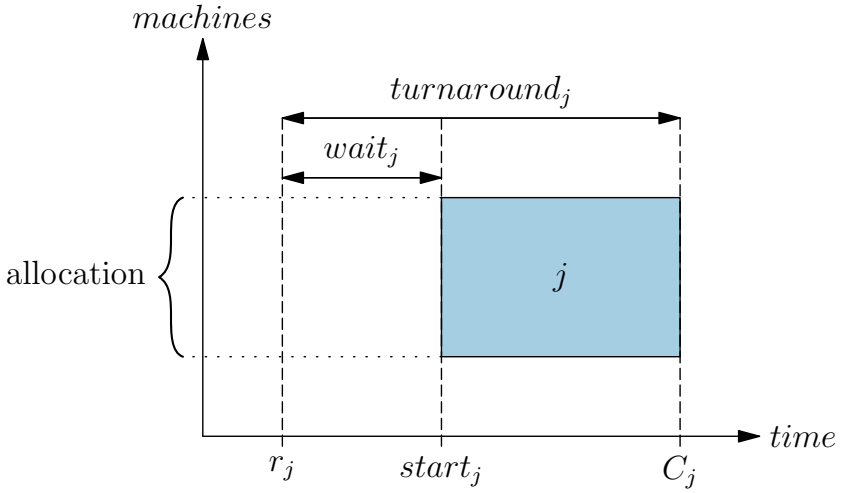


Figure 2.4: Representation of some job-level metrics available after job j completion. The resources allocated to job j are contiguous on the figure, but this is not always the case — as seen on Figure 2.2.

Once the jobs have been executed, more information about them can be defined. We denote by $start_j$ the time at which j starts being executed, and C_j the time at which it completes ($C_j = start_j + p_j$). The job waiting time $wait_j = start_j - r_j$ (in seconds)

denotes the amount of time j stayed in the system before being executed. The job turnaround time $turnaround_j = C_j - r_j = wait_j + p_j$ (in seconds) denotes the total amount of time j stayed in the system. Figure 2.4 summarizes these notations.

The waiting time $wait_j$ can be seen as a user satisfaction metrics, as users most of the time want their jobs to start rapidly. However, users commonly consider different waiting times as acceptable depending on the job duration: Jobs that require a small amount of time should not wait for long, whereas bigger waiting times are tolerable for longer jobs. The job slowdown $slowdown_j$ metrics has been introduced for this purpose [Fei01], is defined in equation 2.1 and is expressed with no unit. The slowdown is sometimes named *stretch* in the literature. However, one can notice that $slowdown_j$ skyrockets if p_j is very small. The bounded slowdown avoids this problem thanks to a processing time threshold τ as seen in equation 2.2.

$$slowdown_j = \frac{turnaround_j}{p_j} \tag{2.1}$$

$$bslowdown_j^{\tau} = \max\left(\frac{turnaround_j}{\max(p_j, \tau)}, 1\right) \tag{2.2}$$

2.2 About the Platform

The platforms studied in this dissertation are composed of computational resources connected via a network of any topology. Both computational and network resources can be heterogeneous. The computational resources may simply be referred to as *machines*.

Machines are indexed by i in a set M. The number of machines is denoted by $m = |M|$. Machines may have multiple *power states*, with a computing speed and a power consumption associated to each power state. Computing speeds are denoted by cs and expressed in number of floating-point operations per second ($flop/s$). Notations may vary within chapters depending on the type of resources whose computing speed or power consumption is given — e.g., a single power state, a machine with one single computation power state or a cluster of homogeneous machines. The power states can model various ACPI states, such as performance states used for Dynamic Voltage and Frequency Scaling (DVFS), processor states which allows to support multiple types of processor idleness, or global states used for powering-off the machine completely or partially — e.g., suspend to RAM. Machines are in one and only one power state at a given time. Switching from one power state

to another may take some time and consume some energy. Precise information and notation about power states is given in section 4.5.

The network can be seen as a graph whose nodes either are machines or network devices (routers, switches...). Nodes are connected with edges (network links). Network devices and links have a latency denoted by lat and expressed in seconds (s), and a bandwidth denoted by bw and expressed in bytes per second (B/s). Depending on the job computation model (cf. section 4.4) network information can either be taken into account to compute the duration of the jobs or be ignored.

We consider that the scheduler has limited information about the platform. The scheduling algorithms used in chapter 4 are oblivious to the platform. In chapter 5, some scheduling heuristics are oblivious to the platform, some others only assume that the order in which the machines are indexed somewhat follows its topology, and finally some heuristics assume that machines are grouped in clusters (non-recursively) and knows the cluster into which each machine is. In chapters 6 and 7 we suppose that the resource management algorithms have an estimation of the power consumption of machines in each state, and an estimation the duration of boot and shutdown operations on each machine.

2.3 Energy

The power consumption is denoted by P and expressed in watts (W). The power consumption of machine i at time t is denoted by $P_i(t)$. The energy consumption of machine i from time t_0 to time t_1 is computed by $\int_{t_0}^{t_1} P_i(t)\ dt$ and is expressed in joules (J).

This paragraph gives the energy model considered for chapters 6 and 7, which only consider homogeneous machines with computation-only jobs. In this case, the machines have various states $S = \{\ computing, idle, off, on \rightarrow off, off \rightarrow on\ \}$. At a given time t, machine i is in one and only one of the states of S. Idle machines can be switched-off, which takes a time $t_{on \rightarrow off}$. Off machines can be switched-on, which takes a time $t_{off \rightarrow on}$. The power consumption of machine i is fully determined by its state. A fixed power consumption is associated to each state. Explicitly, the power consumption of states are denoted by $P_{computing}$, P_{idle}, P_{off}, $P_{on \rightarrow off}$ and $P_{off \rightarrow on}$. Figure 2.5 outlines how the model is instantiated (section 6.5.1 explains how these values have been obtained).

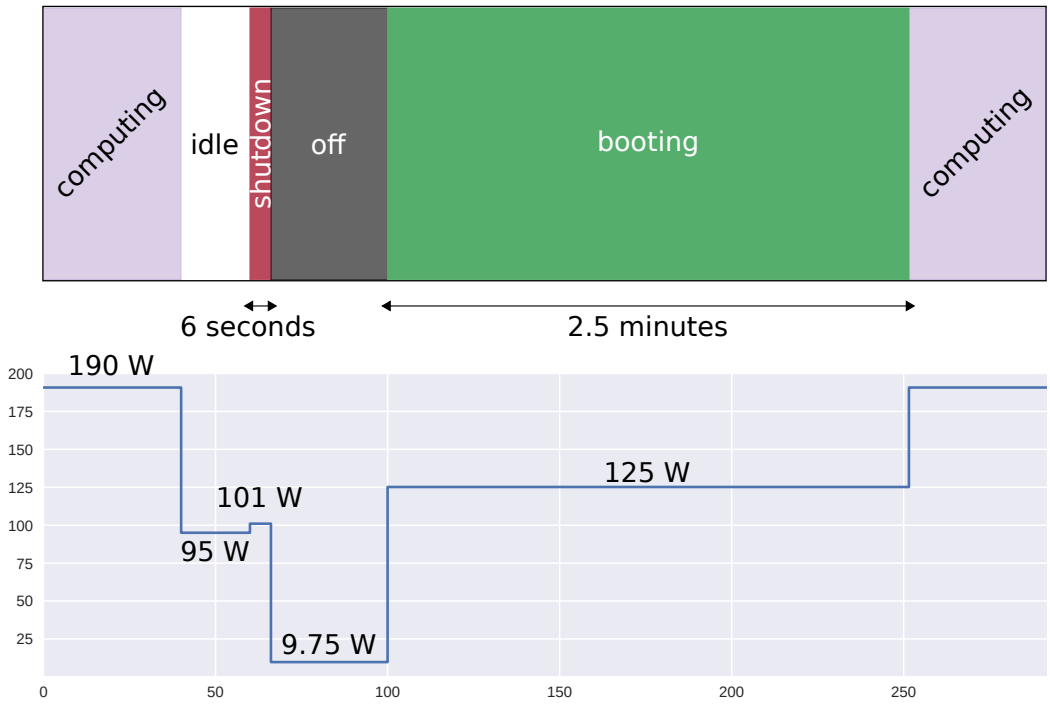


Figure 2.5: Instantiation of the energy model used in chapters 6 and 7. A machine that is computing a job always consumes 190 W. An idle machine always consume 95 W. Switching-on a machine always consumes 125 W for 2.5 minutes — therefore 18750 J.

2.4 Objectives

Different objectives are considered in the dissertation. Chapter 4 and 5 analyzes the makespan, which is denoted by $C_{\max}$ and corresponds to the schedule duration (in seconds). The makespan is defined by $C_{\max} = \max_j C_j$ when the first job is submitted at time 0, or more generally in equation 2.3.

$$C_{\max} = \max_{j \in J}(C_j) - \min_{j \in J}(r_j) \tag{2.3}$$

Chapters 4, 6 and 7 consider user quality of service (QoS) metrics. The first QoS objective is the mean waiting time defined by $\frac{1}{|J|}\sum_j wait_j$. We also consider the mean (bounded) slowdown, which normalize the waiting time by the job processing time. The mean slowdown is defined by $\frac{1}{|J|}\sum_j slowdown_j$, and its bounded counterpart by $\frac{1}{|J|}\sum_j bslowdown_j^{\tau}$ where τ is the processing time bound.

Chapter 7 also consider the maximum counterparts of the previously defined metrics. The maximum waiting time is defined by $\max_j (wait_j)$, the maximum slowdown by $\max_j (slowdown_j)$ and the maximum bounded slowdown by $\max_j (bslowdown_j^\tau)$.

Chapters 6 and 7 consider the total consumed energy. It is defined by the power consumption of the machines during the period of interest as seen on equation 2.4, as only the power consumption of the machines is considered.

$$\sum_i \int_{\min_j r_j}^{\max_j C_j} P_i(t)\ dt \tag{2.4}$$

Finally, chapter 7 considers the total number of power switches. The objective is defined by $\sum_i \#switch_i$, where $\#switch_i$ denotes the total number of power switches (switch-on and switch-off) done on machine i during the whole schedule.

3 Why Batsim?

3.1 Introduction

Research about how to manage large scale distributed computing platforms efficiently has been conducted for dozens of years and remains a highly active domain, which includes a wide range of studies from pure theory to practice. Contemporary platforms may have a tremendous number of computing resources, connected by complex network topologies. Heterogeneity can be found in the computing resources and in the network connecting them.

Many problems arise on current platforms and are mostly multiobjective. For example, platform administrators often want to maximize the utilization of the platform, while users usually desire a good quality of service. More and more constraints must be dealt with today to make the best of the computing platforms. In particular the system energy consumption became a limit to build bigger platforms — but many other problems must be dealt with such as data movement and resilience. Facing all these problems at the same time is very hard, as the problems taken alone are already very complex.

Simulation is the most used technique to study this kind of system, as *in simulo* experiments costs are much smaller than those incurred by *in vivo* ones. For example, simulating the execution of a 1-month workload from a TOP500 supercomputer may only take few minutes on a single middle-class computer, which allows extreme time and energy savings. Furthermore, simulations are most of the time deterministic, which is great for reproducing experiments.

However, as simulation results only include phenomena that are modeled by the simulator, conclusion only based on such results are to be taken with caution for two main reasons. The first reason concerns the simulation models realism. As current computing platforms comprise several complex layers, some hypotheses that are made by naive simulation models do not hold in practice. For example, the hypothesis that jobs take a fixed amount of time regardless of where they have been allocated rarely holds, unless the platform is fully homogeneous and if the network cannot be saturated — or if all applications are insensitive to network contention.

The second reason concerns the simulator implementation. Even if the simulation models accurately outline the desired phenomena, implementing them correctly is tricky. Making sure that the different types of phenomena are correctly simulated together — and making sure that the simulator will continue to produce correct results — involves recurring validations with *in vivo* experiments and good software engineering techniques. The following section describes some pitfalls that we have experienced or observed in the simulators of the field. Most of those pitfalls involve an insufficient separation of concerns.

3.2 Common Simulation Pitfalls

3.2.1 Strong Simulator/Algorithm Coupling

The simulation of the resources is often strongly coupled with the resource management algorithm — or more simply the scheduler. While strongly coupling these two components may simplify the simulator development and allow stronger simulation time optimizations, we think that this is more detrimental than beneficial in most cases.

Most of the time, strong coupling imposes constraints on the algorithms — e.g., to store jobs in queues — and strong implementation constraints — e.g., to use a specific programming language with the supplied API. We think that these constraints lead to a duplication of simulators for poor reasons — e.g., who never thought/said/heard "There is no way I implement my algorithms in [language A], so I will reimplement the whole (simulator, algorithms) couple in [language B]!". In the long run, we think that strong coupling reduces the lifetime of simulators. This issue is all the more important as in this case the simulator death is likely to result in the *loss* of the algorithms, in the meaning that they will not be comparable to new algorithms anymore unless they are reimplemented with another simulator.

Moreover, we think that strong coupling fosters unrealistic simulation practices. As platform information is accessed at no cost in such approaches, one is easily tempted to directly use precise information to take decisions. However, resources are monitored at low frequencies in the real world unless the platform administrators agree to hinder the platform main use case for monitoring purpose — or agree to invest in a dedicated network. This forces the decisions to be taken with partial and delayed information in production. Therefore, this hypothesis makes difficult the adaptation of such algorithms for production use — or even impossible if the

information cannot be acquired at all in practice. We also think that strong coupling tends to reduce the soundness of evaluation results, as the exact same models are more likely to be used in simulation and in the decision-making process.

On the other hand, a loose coupling approach allows many more researchers to use the simulator, as no one would be forced to use any language nor API. This design choice highly decreases the risk of the algorithm implementations to disappear, as even if the simulator is not maintained anymore, modularity ensures that such implementations can be used within other simulators or even in production systems.

As the modular algorithms implementations are less likely to disappear, we think that comparing them to new policies is sounder and easier. In most cases, one need to reimplement non-modular algorithms from the literature if one wants to compare them to one's new policies. However, reimplementing these policies properly may be tough, as implementation details — that are hardly ever included in research papers — may comprise critical information without which the algorithm efficiency may be threatened. Hence, we think that loose coupling may improve the fairness of comparisons between existing and new resource management policies, as the risk of poorly reimplementing existing policies would be avoided.

3.2.2 Restricted Simulation Models

Another pitfall when designing simulators is to only focus on specific simulation models. Section 3.2.1 depicted that a too strong coupling between the simulator and the algorithm could be detrimental. This section takes the separation of concerns reflection further and discusses about the modularity of simulation models.

Most simulators are implemented with a unique simulation model in mind. This model is naive most of the time. As an example, we can think about the numerous scheduling simulators that only operate fixed-length jobs. These simulators are used to observe and assess how theoretical scheduling algorithms behave in practice — in addition to worst-case or average-case analyses/guarantees/proofs. However, as naive models ignore most of the phenomena that may occur on real platforms, one may lack confidence about the realism of such results.

Simulators with multiple simulation models can partly solve this confidence issue. In this case, if the naive simulation model is considered too unrealistic in certain scenarios, a more realistic one could be used instead in these cases. The simulation

model modification can be completely transparent to the resource management algorithm if the simulator is modular enough.

More generally, simulators with interchangeable simulation models allow the users to choose how to simulate each type of phenomenon, thus allowing a customizable level of realism. As more realistic simulation models can be significantly longer to simulate than naive ones, this approach allows to first assess the algorithms quickly, then to study more realistically particular scenarios.

Additionally, simulators that comprises models for different types of phenomena can be very convenient for the users. Novel policies are often proposed to solve very specific problems, but as the real problems are multiobjective and involve multiple types of phenomena, one may be interested in the bigger picture. One may for example propose and evaluate different placement algorithms, considering that the jobs communicate internally and taking the platform topology and parameters into account. As most real jobs read input data and write output data, it would be interesting to know whether the previous results would be the same if I/O phenomena were taken into account. The possibility to toggle on the simulation of I/O in the simulator would be interesting and convenient for the user in this case.

3.2.3 Publish And Perish

Many simulators are developed to reach short-term objectives. They are used to experiment new policies and to write some articles, but they often lapse afterward. The rest of this section details why we think this philosophy should be avoided as far as possible.

The duplication of simulators may at first glance look profitable for research reproducibility, as the results coming from the different simulators can be compared with each other. However, as many simulators do not subsist long enough to be assessed against *in vivo* phenomena, we think that the effect is rather the opposite. Some simulators are never released to the public, others have been lost and their implementation cannot be found anymore. Only limited confidence can be given to results coming from non-validated simulators. Even if the selected simulation models have already been validated, making sure that no embarrassing bug is hidden somewhere in their implementation takes time. Forsaking a simulator quickly after retrieving results from it increases the risk of occurrence of undetected errors in the scientific results, as there is little chance that the simulation output will be looked at ever again.

We think that investing time in a robust simulator is more time-efficient than the common *quick and dirty* approach. A greater initial effort is needed if one wishes to develop a robust and modular simulator, as many factors need to be taken into account to make sure that future evolutions will not incur dramatic changes. However, future users — e.g., new PhD students — will not need to reimplement the whole simulation base again, which will avoid wasting time. Newcomers would then benefit from the existing confidence into the simulator and will improve it, either by conducting validation experiments or simply by using the simulator, as bugs are more likely to be detected if the simulator is heavily used. As the improvements made by the different users can be shared, we think that this approach is beneficial for the quality of results but also for productivity.

Some simulators that falls into the *publish and perish* category model specific phenomena in great details. We think that this design choice lowers the chance of adoption of the proposed simulation models, even if the simulated phenomena interest a large proportion of the community. Some phenomena are very specific and may be hard to integrate into general purpose simulation frameworks, but we think that the vast majority of phenomena can be included in such frameworks. This would allow many researchers to use these results, to (in)validate them and therefore would help understanding the phenomena.

3.3 Goals

Batsim main goals are shared with most of the simulators of the field: To allow the comparison of different resource management policies, in a reproducible and efficient manner. However, Batsim has been designed with many other goals in mind. This section aims at describing them.

We realized that the algorithms coming from theory differ greatly from those implemented in production systems. This gap between theory and practice has multiple causes, which are partly caricatured in the following paragraph. Theoretical researchers usually do not bother to implement their algorithms in production systems, as making production-compatible algorithm takes a considerable amount of time, which would not be used to produce scientific results. On the other side, engineers operate complex production systems that include many features. As some theoretical algorithm are based on models far from reality or may not look compatible with existing features, engineers might not see what they would gain by including novel policies in the systems they manage. One consequence of this situation is that the

scheduling algorithms in production systems are mostly outdated, while they would probably benefit from state-of-the-art management policies.

Reducing the gap between theory and practice is one Batsim goal. Batsim design forces the modularity of the decision making procedures, by placing them into separate modules that can be implemented in any programming language. The decoupling obviously allows the management algorithms to be used with Batsim, but also allows them to be connected to other simulators or to be integrated in production systems. Consequently, this approach improves the production launch of algorithms, as the algorithms that are first developed and analyzed by researchers can then be used within real systems as is. The modularity can also be used the other way around: Existing production algorithms can be connected to Batsim, which allows the exploration of many scenarios in simulation. This type of connection eases the parameter tuning of production algorithms, as *in simulo* exploration is faster and cheaper.

Another Batsim goal is to be customizable regarding simulation models. Our main focus is to allow multiple levels of realism about how the jobs are computed, from naive fixed-length black boxes to the replay of MPI applications, passing by time-agnostic communication matrices. We also want to take as much phenomena as possible into account, while letting the user choose which phenomena should be simulated. The idea behind this approach is to allow a wide area of experiments thanks to the simulator, and to be modular enough so that other phenomena can be annexed to it later on. For example, we want to be able to simulate the behaviour of different computing platforms, with different job models including network or I/O contention, while simulating how much energy the platform consumes.

Finally, we want Batsim results to be reliable. To do so, we first chose to base Batsim upon SimGrid rather than implementing the simulator from scratch. SimGrid is a state-of-the-art simulation toolkit that allows to study applications in distributed environments. This choice increases the confidence we have in Batsim results, as SimGrid is a long-term project that contains deeply validated models. This choice also increases Batsim life expectancy, as Batsim will benefit from the improvements done within SimGrid, such as corrections in existing models or the addition of simulated phenomena. We also want to invest time in Batsim implementation, notably by using regression tests and continuous integration to make sure existing features will continue to function as expected. Batsim and SimGrid are fully open source, which avoids the risk of *losing* the work conducted and increases confidence and reliability — as anyone is free to explore and improve the implementations.

4 Batsim: a Realistic Language-Independent RJMS Simulator

4.1 Related Work

Many scheduling algorithms are assessed by simulation in the literature. Unfortunately, most of the simulators used are never released to the public. Many reasons can be suggested to explain this. The authors may just not want to maintain released software, as it is time-consuming. The simulator implementation may also have been done the *quick and dirty* way, and the authors do not want anyone to look at the code they produced. Regardless of the authors rationale, we think that this causes substantial reproducibility problems, as even merely trying to repeat the conducted simulation processes requires a lot of investment. This related work section is about existing simulators that can be used to evaluate jobs and resources management systems. To be precise, we chose to only consider simulators interested in the realism of the produced results and whose source code can be found. This last criterion drastically reduces the number of candidates, as many simulators either have never been released to the public or fell into disuse and cannot be found anymore.

To the best of our knowledge, most scheduling simulators are either very domain specific — e.g., Realtss — or do not primarily focus on realism of results, as comparisons to real existing systems are hardly ever done. This can be explained by the financial and ecological cost of such evaluations, but this is quite hazardous as models may be irrelevant in unevaluated scenarios.

Alea [KR10] is probably the closest approach to ours. This simulator is based on the GridSim simulation toolkit and allows to compare different scheduling algorithms. Both Alea and Batsim chose to rely on existing simulation frameworks instead of rebuilding everything from scratch. Alea however does not essentially focuses on separation of concerns, which does not allow straightforward comparisons to real RJMS code, nor allow freedom regarding programming language. At the best of our knowledge, this simulator has not been validated in a real environment yet.

Another interesting approach can be found in article [PML15]. This approach consists in using the INSEE [PM05] fine-grained network simulator offline to obtain very precise execution time in all possible job configurations. Article [PML15] proposes job placement policies that guarantee that no network interference can occur between the jobs, allowing to use offline execution times while simulating an online workload. This approach is realistic in the studied scenario but cannot be used when jobs may interfere, which happens in wider scenarios.

A previous initiative of building a scheduling simulator on top of SimGrid has been done in Simbatch [CG09]. The project was unmaintained from 2007 to 2015, but has recently been updated so that up-to-date SimGrid versions can be used with it. As Simbatch has not been implemented with separation of concerns in mind, it cannot be used in conjunction with production code nor with algorithms written in any language.

Some existing RJMSs can be used in simulation. This is for example the case for the Flux RJMS [Ahn+14], in which a simulation mode is included. Slurm [YJG03] has not been designed with simulation in mind but work has been conducted to allow the tuning of parameters in simulation [Luc11]. Such simulators use naive models and do not focus on simulation realism but on the assessment of real code in simulation. There are therefore complementary to ours. A Batsim adapter could for example be incorporated in the work conducted in [Luc11] to use Slurm's code with Batsim — just as we did for the OAR RJMS [Cap+05] in the present chapter.

4.2 Batsim General Description

Batsim is an open source[1] RJMS simulator. It allows to simulate the behavior of a computational platform on which workloads are executed according to the rules of a scheduling algorithm. As depicted in chapter 3, Batsim promotes separation of concerns and thence is itself built on top of the SimGrid simulation framework [Cas+14]. This choice benefits Batsim, as it widely broadens the scope of the possible experiments, increases the simulation results soundness and will allow Batsim to profit from future SimGrid features or improvements.

Batsim aims at improving practice in the implementation of resource management algorithms. For this purpose, separation of concerns is applied on the RJMS itself.

[1] Batsim is distributed under the LGPL-3.0 license. The project source code and documentation are available on Github [@batgit1] and on Inria's Gitlab [@batgit2]. Release versions can directly be used with the oarteam/batsim docker container.

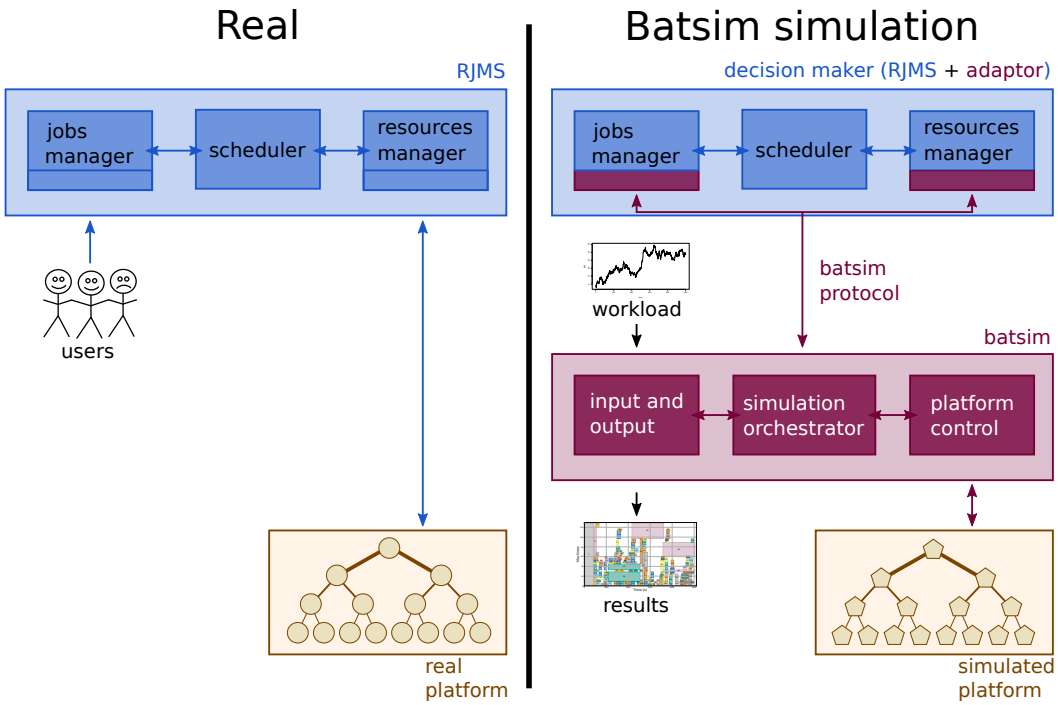


Figure 4.1: Batsim simulation overview.
Simulation logic is split into two components. Batsim orchestrates the simulation, manages the platform and handles inputs and outputs. The other component is in charge of making most decisions and may therefore include parts from real RJMSs. This figure does not detail how Batsim works internally.

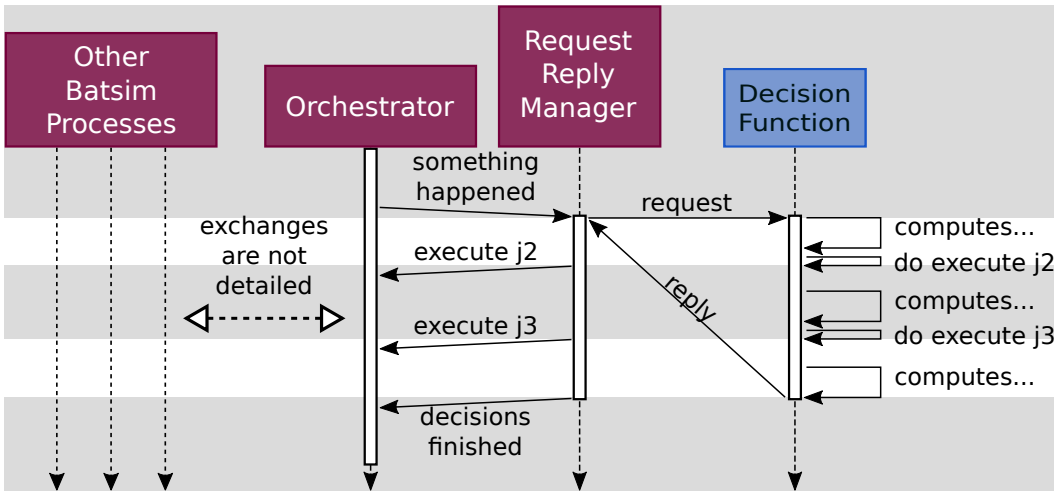


Figure 4.2: Portion of a Batsim simulation sequence diagram.
The Batsim component is composed of concurrent *processes* that communicate with each other. In this example, the decision-making component makes decisions that should be injected on-the-fly — i.e., it first takes time to think, decides to execute job 2, thinks again, decides to execute job 3, thinks again then finally finishes. Batsim dictates how the simulation time progresses. As soon as the decisions are received, Batsim injects them into the simulation at the specified times. Concurrent *calls* to the decision function are forbidden.

A typical Batsim simulation involves two components, as seen on figure 4.1. The **Batsim** component orchestrates the simulation and manages the computational resources, while the **decision-making** component makes decisions — as the name suggests. The two components interact through an event-based protocol.

This clear separation allows to use decision making procedures coming from real RJMSs in simulation, as seen on figure 4.1. To do so, the RJMS must implement a small adaptation layer to *mock* the regular platform communication layer — orders must be sent through the Batsim protocol and not directly applied on the resources, and reading the events received from the Batsim protocol must be done instead of the usual monitoring of the platform. Batsim therefore eases the modification and the tuning of real RJMS codes, as such changes can be tested easily *in simulo*.

Please note that this adaptation can also be done the other way around, such that any Batsim-compatible decision-making component can be connected to a real RJMS. This other type of adaptation improves the production launch of new algorithms, as Batsim-compatible decision making procedures can easily be used in production. It also improves the algorithms life expectancy, as modularity allows them to be connected to other simulators or real systems. In this case, the small adaptation layer to apply to real RJMSs consist in forwarding the events through the Batsim protocol and to apply the received decisions, instead of calling the regular RJMS decision-making layer.

Another asset of this separation is that the decision-making procedures can be implemented in any programming language. This may look like a detail, but we think that freedom regarding programming languages drastically reduces the likelihood of unneeded reimplementations. It allows to use almost any existing event-based resource and jobs management code thanks to an adaptation layer — whose cost may vary depending on the existing code modularity. In addition, this feature allows newcomers to use the simulator and to compare their algorithms to the existing ones, even if they only wear by rare, obscure or esoteric languages.

Batsim simulates what happens on the platform and *calls* the decision-making component as soon as interesting events are captured — on condition that the component is available. As I write these lines, the two components are instantiated as processes — within the Operating System processes meaning — and communicate through a ZeroMQ [Hin13] socket. Messages are formatted in JSON. Details about the messages content can be found in the Batsim protocol documentation [@batproto]. The *call* to the decision-making component is in fact a network request done thanks to the common request-reply pattern. The simulation is stopped while Batsim waits for the decision-making component reply. Once the reply is received, the decisions

are injected in the simulation at the specified times. The decisions are mostly about the execution or rejection of jobs, or about the management of machines — e.g., switching-on some machines, switching-off them or changing their DVFS state. The protocol also includes other features, such as calling the decision-making component at a specified time or the dynamic injection of jobs in the simulation. Please refer to the Batsim protocol documentation [@batproto] for a complete list of the available decisions and the associated information.

As real decision-making procedures take time and are not always interruptible, Batsim implements a generic way to take into account the times at which the decisions are taken. This allows many scenarios: 1. the decision-making time can be ignored 2. the decisions can be applied after the decision-making procedure 3. the decisions can be applied when they are taken 4. the decision maker can take advantage of this to avoid being called prematurely. Batsim dictates how the simulation time progresses. When a *call* to the decision-making component is conducted, Batsim tells the current time t_c and the times t_e at which each event e occurred. The events are either in the past or the present: $\forall e, t_e \leq t_c$. The reply consists in the time t_r at which the decision-making component replied and in a series of decisions — with a time t_d associated to each decision d. These times allow several ways to take the decision-making time into account.

This paragraph details how Batsim manages time dilation, as the two components are temporarily in different time lines when a decision-making *call* is realized. First, the decision-making time can be completely ignored if all the replied times are done at the present time: $t_r = t_c$ and $\forall d, t_d = t_c$. Second, the decisions can be applied when the decision-making procedure completes. To do so, the reply is sent in the future and all the decisions are made at the same time: $t_r > t_c$ and $\forall d, t_d = t_r$. Third, the decisions can be applied on-the-fly, as seen on figure 4.2. To do so, the decisions are made in the future at different times: $t_r > t_c$ and $\forall d, t_d \in [t_c, t_r]$. In all cases, please note that time consistency is enforced by forbidding concurrent calls to the decision-making procedures. This implies that the decision-making component may not be notified of events as soon as they occur, but only once the component is available — just as it would happen on a centralized real RJMS.

4.3 Batsim and SimGrid

As Batsim is built on top of SimGrid [Cas+14], a brief and simplified overview of SimGrid is conducted in this section. This section also enumerates the small

differences that exist between Batsim and SimGrid, which may be useful for readers already familiar with SimGrid. Please notice that this section only describes the MSG SimGrid API and that other vocabulary and models might be more relevant if other SimGrid APIs were considered. A **host** is a resource that can compute floating-point operations (flop). According to the notations of chapter 2.2, a host corresponds to a machine. Hosts are connected via a network whose **links** have a latency (in seconds) and a bandwidth (in bytes per second). Links are hyperedges that can either represent a network node — a switch or a router — or a network link — a connection between two nodes. Hosts and links compose a platform that can be of virtually any topology.

SimGrid processes are executed on hosts. At a given time, a SimGrid process is on one and only one host. SimGrid processes are user-given source code executed within the simulation. They can execute arbitrary instructions — that do not consume simulation time — or call functions from the SimGrid API that cause simulation time increases. For example, computing a *task*, sending or receiving *messages* from/to another SimGrid process, spawning another SimGrid processes or just sleeping are time-consuming operations.

A SimGrid-based simulator is therefore a set of SimGrid processes that compute user-given functions. SimGrid orchestrates how the functions are executed by executing one function at a time. To be precise, SimGrid executes the current function as long as possible, and stops when an instruction that impacts the simulation time is found. SimGrid then solves which function should be called next, at which time, and how the simulated operations progress — notably computations and network transfers.

The resources studied in a SimGrid simulation are described in a platform file. Batsim uses exactly the same platform format. However, Batsim adds little constraints such that platforms may require a little adjustment to work with Batsim. Batsim uses a set of hosts to compute the jobs, while other hosts are reserved for specific operations. For example, most of the resource-management SimGrid processes are executed on a host called the **master host**. The platform must allow two-ways communication between the master host and the computation hosts. Additionally, jobs that require internal communications can only be allocated on hosts that can reach each other — in the network route meaning. Other specific hosts may be needed depending on what one wants to simulate, such as hosts for managing a parallel file system. Finally, as described in section 4.5, Batsim platforms may require more information than regular SimGrid platforms when one wants to simulate the platform energy consumption.

4.4 Job Computation Models

Batsim workloads are divided into a set of jobs and a set of simulation profiles. While jobs describe user-level job information, the profiles describe how the jobs are simulated. A job essentially consists in the user request and in the associated simulation profile. Explicitly, each job j has a unique identifier id_j, a release time r_j (the time at which the job is submitted into the system), an optional wall-time $wall_j$ (the user-specified execution time bound such that j is killed if its execution time exceeds $wall_j$), a number of requested resources q_j and the profile $prof_j$ the job executes. As the Batsim workload format is extensible, please do note that more information can be included in the jobs depending on the users' needs — and that this information is forwarded to the decision-making component.

Separating the simulation profile from the jobs avoids data duplication when many jobs are computed in the same way, and makes workload generation easier and more modular. Different types of jobs profiles are available in Batsim and correspond to different job computation models.

To start with, the **delay** profile type only consists in a fixed amount of time. A host executing this profile type only sleeps for the specified amount of time. Such profiles completely ignore the job execution context and do not lead the reserved hosts to use more energy than when being idle, as no *work* is computed on the hosts. They are however easy to instantiate as most traces contain the jobs duration.

The **parallel task** profile type can be used if one desires to take the the job execution context into account or is interested in energy consumption. Such tasks combine computations and communications and are executed on a set of hosts. To be precise, they are defined by a computation vector $comp$ and a communication matrix $comm$. Each $comp_k$ represents the amount of computation (in flop) that must be computed on the k^{th} host. Each element $comm[s, d]$ of the communication matrix represents the amount of communication (in bytes) to achieve from the s^{th} host to the d^{th} one. All the sub-operations encapsulated into a parallel task — computations and point-to-point communications — are strongly connected. The execution of a parallel task can be seen as a *cursor* that starts at 0 and finishes at 1. That cursor sets the already achieved fraction of all the sub-operations. Let us for example consider a parallel task that only includes two computations of 10 and 20 flops. At the middle of the task execution, it is mandatory that 5 flops have been computed on the first host and that 10 flops have been computed on the second one. Please do note that the rate at which the cursor progresses depends on the execution context, as SimGrid updates it to match the current bottleneck. Parallel tasks models jobs in coarse grain,

as the way they are executed supposes that the underlying applications are executed smoothly.

SMPI traces can be used if one is interested in fine-grain simulation. Such profiles consists in replaying an existing SimGrid time-independent trace. The trace contains a suite of operations for each host, which are either computations (in flop) or MPI function calls. Such profiles therefore define some dependencies between the operations, which allow to observe more precise phenomena than parallel tasks. For example, we can observe that slowing down an application during a critical phase leads to higher time increases than in a non-critical phase. However, finer-grain simulations take longer than coarse-grain ones.

As I write these lines, profiles can be composed sequentially but not in parallel yet. Other profile type exist such as wrappers to build homogeneous parallel tasks or to operate parallel file system operations. Please refer to our implementation [@batgit1] for an exhaustive list of the existing profile types.

4.5 A Few Words About Energy Simulation

SimGrid computes the platform energy consumption. This paragraph details how the energy consumption is simulated. Please note that this model only considers the hosts energy consumption at the moment. Each host h has a set of power states denoted by PS_h. Each power state $p \in PS_h$ has a computational power cs_p (in flop/s), a minimum electrical power consumption $P_p^{\perp}$ (in W) and a maximum electrical power consumption $P_p^{\top}$ (in W). During its usual simulation process, SimGrid computes the *load* of each host — and of each link. The load of host h at time t is a real number in [0,1] denoted by $l_h(t)$, where 0 represents an idle host and 1 a host fully computing[2] . If we denote by $p_h(t)$ the power state in which host h is at time t, the instantaneous electrical power consumption of the host is fully determined by $p_h(t)$ and $l_h(t)$. This instantaneous electrical power consumption is denoted by $P_h(t)$ and expressed in watts. It is computed as as the linear interpolation between $P^{\perp}_{p_h(t)}$ and $P^{\max}_{p_h(t)}$ in function of $l_h(t)$. Explicitly, $P_h(t) = P^{\perp}_{p_h(t)} + (P^{\top}_{p_h(t)} - P^{\perp}_{p_h(t)}) \cdot l_h(t)$. The energy consumption of host h is therefore given by $E_h = \int P_h(t)\ dt$ and expressed in joules.

[2] In SMPI, hosts either compute something at full speed or do not compute anything, which leads to a binary load. However, when parallel tasks are used, the *cursor* constraint (cf. section 4.4) may set the hosts load to any value in [0,1].

Batsim adds a slight layer on top of the SimGrid energy consumption model. We chose to split the set P_h of the power states of the host h into three disjoint sets. P_h^c is the set of **computation** power states, P_h^s is the set of **sleep** power states and P_h^t is the set of **transition** power states. The computation power states are the only ones that can be used to compute jobs. A sleep power state represents the state of a machine that cannot directly compute something *e.g.* ACPI S1, S3, S4 or S5 states. A Batsim host can switch from one computation power state to another instantaneously. However, entering into a sleep power state s or leaving it can take time and cost energy. Transition power states are *virtual* power states that are only used to simulate the transition into and from sleep power states. To do so, costly transition are simulated by 1-flop computations. If transition t should take time t_t (in seconds) and consume e_t energy (in joules), the corresponding virtual power state p_t should have a computational speed $cs_{p_t} = \frac{1}{t_t}$ and an electrical power consumption $P_{p_t}^{\perp} = P_{p_t}^{\top} = \frac{e_t}{t_t}$.

If one desires to simulate the energy consumption with Batsim, the platform file to use must define the type of the hosts power states — in addition to fulfill the requirements of regular SimGrid energy platforms. Explicitly, the user must define for each sleep power state $s_h \in P_h^s$ the transition power state $p_{s_h}^{\downarrow}$ used to switch into s_h, and the transition power state $p_{s_h}^{\uparrow}$ used to leave s_h. All other existing power states — those that have not been marked as sleep or transition ones — are considered as computation power states.

4.6 Implementation Details

Batsim is a C++ program that uses the C MSG SimGrid API. As described in section 4.3, Batsim is essentially a set of concurrent processes that communicate with each other thanks to messages, whose execution is managed by SimGrid. This section sketches how Batsim works internally, by describing the main SimGrid processes and how they interact. This architecture is more difficult to apprehend than a *simple* event loop, as the simulator is itself a distributed application. It is however very convenient to describe complex operations and allows a great separation of concerns. For example, if one is not satisfied with the current simple job launching mechanism and desires a more detailed one instead — e.g., including an authentication procedure or data movement — one could implement it without precise knowledge of any other simulation part. In this case, one would only need to implement a new SimGrid process with similar *external* behavior than the current job launching SimGrid process, and to spawn one's new process instead of the current

one. The external behavior of such process is essentially how it communicates with the orchestrator — which is introduced in the next paragraph.

The **orchestrator** can be seen as the main SimGrid process. It is unique and spawned on the master host at the beginning of the simulation. Its role is to orchestrate the simulation — as the name says. Most of the other SimGrid processes are spawned by the orchestrator. The orchestrator is aware of all the events that may lead to a decision-making *call*, and manages when to *call* the decision-making component. It is also in charge of injecting the decisions made by the decision-making component in the simulation — e.g., perform a job launching when ordered to. The orchestrator is implemented as a loop that reads messages — from most other SimGrid processes — and reacts accordingly. The orchestrator stops to read messages when it detects that the simulation should stop, which effectively end the simulation[3].

The **workload injector** is in charge of reading a given Batsim workload and of submitting the corresponding jobs at the right simulation times. To do so, it initially parses the workload file, then simply iterates over the jobs in ascending submission time order. It sends a message to the orchestrator when the submission time is reached. If the next job submission time is strictly after the current simulation time, the process enters a sleep phase until the aforementioned job should be submitted. The workload injector stops when all the jobs of the workload have been injected. As Batsim allows to simulate the interactions of several workload in the same simulation, multiple workload injectors can be instantiated. One SimGrid process is spawned on the master host at the beginning of the simulation for each Batsim input workload.

The **request reply manager** is in charge of doing a decision-making *call*. Please notice that the decision-making component is in another *real* process — in the operating system meaning. This SimGrid process has three main roles. The first role is to translate the events received from the orchestrator into a valid Batsim protocol message. The second role is to exchange messages with the decision-making component. As I write these lines, this communication is done through a ZeroMQ socket, which means the SimGrid process first sends a *request* ZeroMQ message then reads a *reply* ZeroMQ message. Finally, the last role is to parse the received ZeroMQ message and to send a message to the orchestrator for each decision at the right time — as section 4.2 and figure 4.2 describe. This means that the SimGrid process enters into sleep phases if needed. The process finally sends a message to the orchestrator to announce that the decision-making process can be *called* again, then finishes. This SimGrid process is spawned on the master host whenever the

[3] A SimGrid simulation stops when all the SimGrid processes have finished.

orchestrator decides to *call* the decision-making process. The orchestrator ensures that at any time, at most one request reply manager is being executed.

The Batsim protocol allows the decision-making component to request calls at specific simulation times. This procedure allows many scenarios, such as periodic *calls* to the decision-making component or more complex ones. The **ulterior call manager** SimGrid process role is to allow this procedure. This process is spawned on the master host by the orchestrator as soon as the related decision is received. Multiple instances of this process can exist at the same time. The process simply waits for the needed amount of time, then tells the orchestrator that the decision-making component should be called, and finally finishes.

The Batsim protocol allows to change the power states of the hosts. Switching a host from one computation power state to another is directly realized by the orchestrator. However, boot and shutdown operations are realized by an **on/off switcher** SimGrid process. Such a process is spawned by the orchestrator on every host whose state is requested to change. As described in section 4.5, these operations can take time and consume energy. Such a process therefore first switches the host into a transition power state, second computes a 1-flop task, then switches instantaneously into the requested power state, and finally notifies the orchestrator before finishing.

Finally, the **job launcher** SimGrid process is in charge of launching a job and of executing it on a set of hosts. The Batsim protocol allows the decision-making process to execute a job on a specific allocation A. The size of A may vary depending on the simulation profile, which allows some kind of moldability. A job launcher is spawned by the orchestrator for each job when the order to execute it is received. It is executed on one host in A — in practice, the first host of A. Such a process essentially computes the job according to its profile as discussed in section 4.4. It then notifies the orchestrator either of the job completion or the job failure, and finally finishes.

4.7 Batsim Evaluation Experiment

We set up an experiment to compare Batsim to OAR [Cap+05] in order to evaluate whether Batsim behavior is close to the one of a real RJMS. OAR is a RJMS notably known for being used in the Grid'5000 [Bal+12] infrastructure. We chose OAR over other RJMSs — e.g., Slurm — because OAR's modular design already decouples the scheduling component from the others parts of the system. Therefore, implementing the small Batsim protocol adaptation layer in OAR is rather straightforward.

We tested Kamelot [@kamelot], a conservative backfilling scheduling algorithm implemented in OAR by executing in two scenarios with the same workloads. In the first one, real OAR-managed machines on Grid'5000 are used. In the second scenario, Kamelot is connected to Batsim thanks to a little adaptation layer named Bataar [@bataar]. In addition to evaluate the soundness of the Batsim results, this experiment demonstrate that the Batsim architecture can be used to test production schedulers.

Our experimental process can be split in two major parts. The first part is about workload generation, which is not a detail as one may need realistic job descriptions to obtain a realistic simulation. The other part is about the workload execution in both scenarios and the comparison of the resulting schedules. The first major part requires real programs, their instrumentation and a methodology to create simulation profiles in different job models. This is detailed in section 4.7.1. Subsequently, the way the jobs are put together to form workloads is explained in section 4.7.2. The second major part is described in sections 4.7.3 and 4.7.3.

Using a realistic simulator also means using a realistic simulated platform. Fortunately, the Graphene cluster of the Nancy Grid'5000 site [@nancyG5K] has already been calibrated. Consequently, we chose to use this cluster for our real and simulated experiments. All our real experiments have been done on the Grid'5000 Graphene cluster, reserving the nodes below one switch each time — to avoid external noise from other Graphene users.

4.7.1 Profile Generation

As described in section 4.7, our experimental process includes a workload generation part. The jobs in our workloads must fulfill some requirements to be executed both in a real platform and in simulation. Batsim allows different levels of realism depending on the desired job models, which makes the workload generation process more complex. For example, the parallel task model needs realistic computation vectors and communication matrices to make sense. Furthermore, instantiating the smpi model requires traces from MPI applications.

In order to obtain realistic values for our profile models, we chose to execute real jobs from the MPI version of the NAS Parallel Benchmarks (NPB)[NAS16]. We instrumented the three benchmarks IS, FT and LU to obtain execution traces. The three benchmarks have been compiled and executed for all available processor sizes — powers of two from 1 to 32 — and for tiny to medium data sizes — B to D

depending on the benchmark. Considering NPB limitations, we were able to compile 47 different MPI programs.

Each application has been executed alone on the platform many times, in order to obtain the real execution time of each application — without the instrumentation noise. As the programs have stable execution times, this allowed us to directly generate delay profiles.

We then instrumented the jobs using Extrae [@extrae] to obtain precise — and heavy — execution traces. This trace has been translated to a format that SimGrid understands — a time-independent trace — thanks to a script courtesy of Lucas SCHNORR [@extrea2tit]. Unfortunately, the conversion script was not able to capture all the MPI messages at the time this experiment has been conducted. This added a profile calibration phase in our experimental process.

As SimGrid does not allow dynamic SMPI applications yet, we were not able to evaluate the SMPI profiles in the present chapter. On the other hand, we aggregated the time-independent traces into computation vectors and communication matrices to generate parallel task profiles.

4.7.2 Workload Generation

The workload generation algorithm that we used is described in this paragraph. It is greatly inspired by chapter 9.6 of book [Fei15]. Please note that our workload generation method is not intended to be sound for comparing scheduling heuristics, but only to evaluate how Batsim behaves compared to a real RJMS. The algorithm generates $N = 800$ jobs iteratively. The interarrival submission times of the jobs is computed randomly with a Weibull distribution of shape parameter $k = 2$ and scale parameter $\lambda = 15$. Since the job sizes (the rigid number of resources a job requests) of the real jobs at our disposal are powers of 2 (from 1 to 32), the size of each job is computed with the formula $2^{\lfloor u \rfloor}$ where u is a lognormal variate of parameters $\mu = 0.25$ and $\sigma = 0.5$. Only variates such that $\lfloor u \rfloor$ is in [0, $log_2(32) = 5$] are used to match the sizes at our disposal. The generation of those workloads depends on a random seed, simply referred to as *seed* in the remainder of this chapter. We chose to generate nine different workloads.

4.7.3 Real Workload Execution

Each workload has been executed twice in a real scenario, under two different switches of the Graphene cluster. We chose to use two different switches of machines to obtain more representative results, as even if the machines are theoretically fully homogeneous, they in practice suffer from small differences. We used the reproducible methodology described in section 4.9 to execute the workloads we generated on Graphene. This methodology includes the installation and the configuration of OAR within the nodes we reserved in Graphene. We configured OAR such that it uses the Kamelot scheduler. We also implemented a replay tool that reads a Batsim workload and launches real OAR job submissions at the times dictated by the workload. The OAR submissions launch the MPI programs that were previously generated.

4.7.4 Simulated Workload Execution

Executing the workloads in simulation simply consists in running Batsim and the Kamelot scheduler on the aforementioned calibrated platform file. As described in section 4.7, Kamelot is executed through the Bataar adapter.

4.8 Results

The nine different workloads we generated have been executed twice on a real platform (on identical machines and network, but not on the exact same machines), and twice in simulation (with delay and parallel task profile types). This section presents the different results and analyzes them. Please note that *ptask* is used instead of parallel task in the following figures.

An overview of the execution of all the workloads can be found on figure 4.3. This figure first of all shows that simulation results are close to real ones — considering the makespan and mean bounded slowdown metrics. Indeed, for each workload, the difference between two real results is of the same order of magnitude than the difference between a real result and a simulated one. This figure also shows that the delay and parallel task simulation results are very close to each other. Contrary to delay profiles, the execution time of a parallel task profile depends on the execution context. The closeness of the two job models can be explained by the high platform homogeneity and the lack of contention in this experiment.

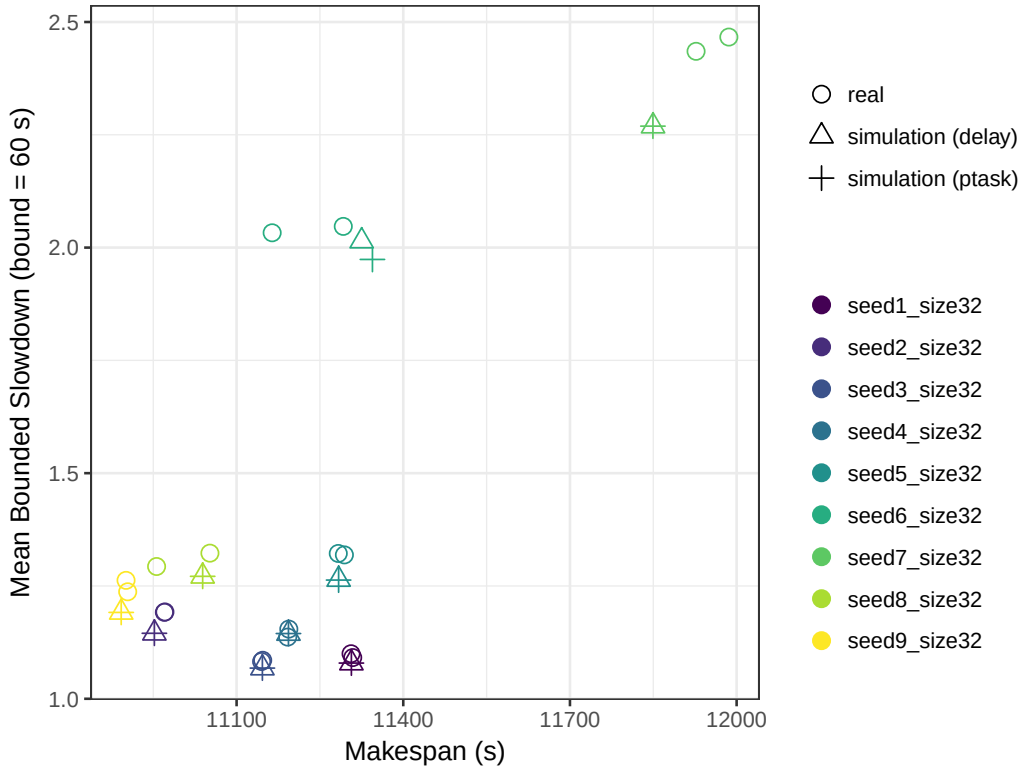


Figure 4.3: Mean bounded slowdown against the makespan of all workload executions. Each point represents one workload execution. Circles are executions on the real platform, triangles and crosses are simulated executions with respectively delay and parallel task profiles. Each workload is associated with one color.

Figures 4.5 and 4.6 shows the differences in mean slowdown between real and simulated executions of all the workloads. The mean slowdown of real executions is in range $[1.492, 5.876]$. These two figures shows that Batsim slightly underestimates the mean slowdown for most workloads. This is in fact explained by a clear underestimation of the jobs waiting times as seen on figures 4.4 and 4.7 — as the slowdown is basically a normalized waiting time.

The Batsim waiting time underestimation — in comparison with OAR — can be explained by the SSH job launching procedure used by OAR. As Batsim tries to simulate the behavior of any RJMS and does not specifically focuses OAR, we preferred to implement a more simple job launching mechanism. Overfitting OAR's job launching mechanism can be done by changing the job launching SimGrid process, as explained in the beginning of section 4.6.

We can finally notice that differences exist between simulated and real schedules if look at them with a finer grain. For example, figure 4.8 shows that the Gantt charts

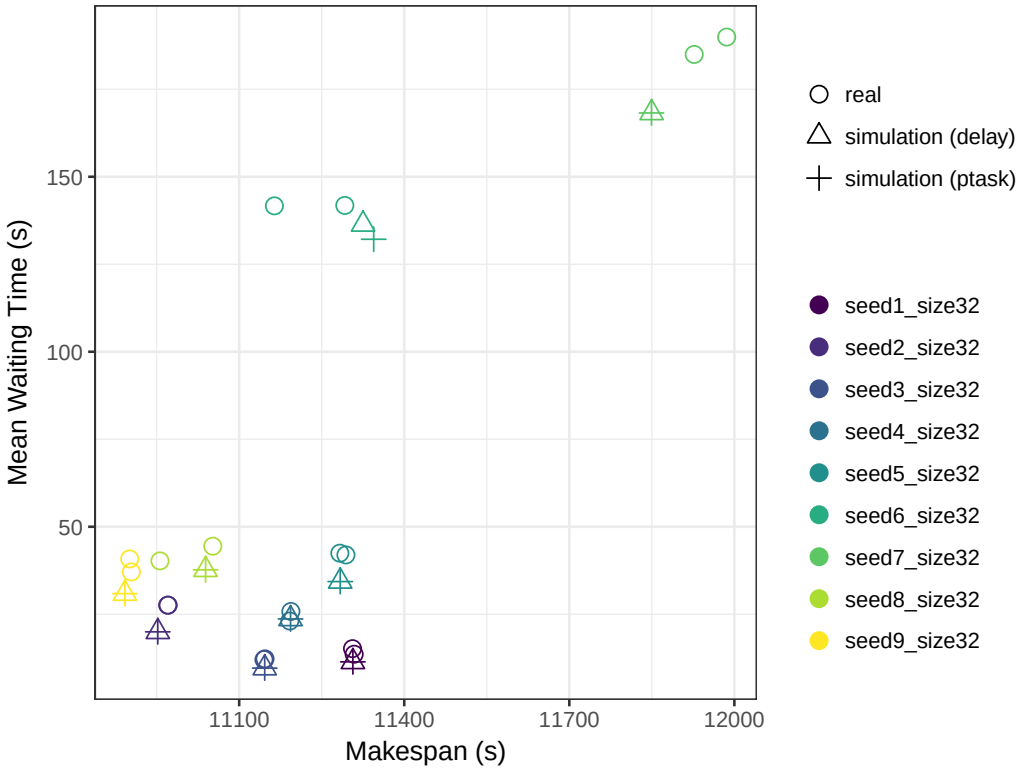


Figure 4.4: Mean waiting time against the makespan of all workload executions. Each point represents one workload execution. Circles are executions on the real platform, triangles and crosses are simulated executions with respectively Delay and MSG profiles. Each workload is associated with one color.

of real and simulated executions of the same workload differ. However, this figure also shows that the same differences can be observed between two real executions.

4.9 Reproducing our Work

As said in chapter 3, one of the main Batsim goals is to foster reproducibility in the field of jobs and resources scheduling, by providing the tools needed to make more reproducible science. The aim of this section is to explain how to reproduce most of our evaluation process. To this end, we provide a complete environment to use Batsim and the different schedulers that run on top of it. All the experiment tools mentioned in the remaining of this section (Batsim, Kameleon, Execo, Grid'5000) are necessary to repeat the experiments we have conducted.

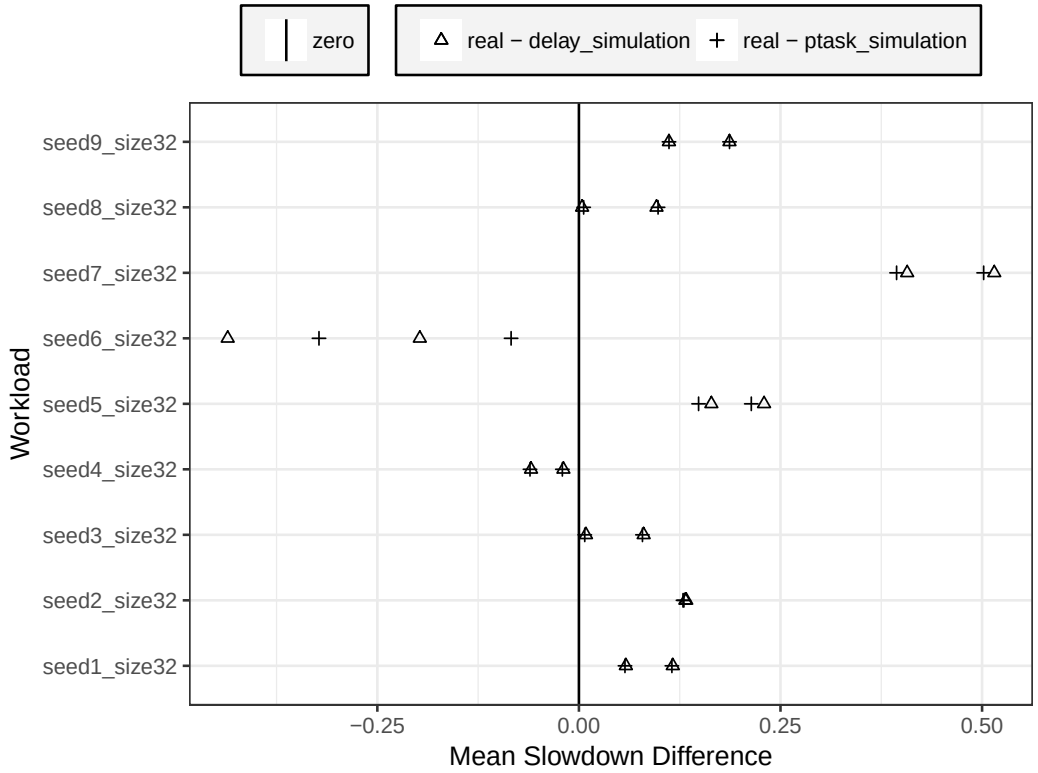


Figure 4.5: Mean slowdown difference between real and simulated executions of each workload. Different workloads are plotted on different lines. Triangles and crosses represent the difference from the simulated execution with respectively delay and parallel task profiles.

Environments An environment can be seen as an object that fixes a software context. It typically includes an operating system and a set of programs and libraries, specifying which version is used for each component. We used Kameleon [Rui+15] to describe and build our environments. Kameleon allows to build environments from template recipes and to extend them. Kameleon's main advantages are the management of different execution contexts — e.g., inside the environment to build, or in the external build environment — and its breakpoint mechanism. The environment produced by Kameleon can be exported to many formats, including virtual machines, docker containers or tarballs, which allows deployments almost anywhere. The Batsim complete environment and the workload generation environment recipes are both available in the simctn Git repository [@batctn].

Experiment design and workflow Most of the time, the experiment design consists in one or more documents that describe the purpose and the experiment with some details and some dedicated scripts. Some domain specific tools exist to compute the

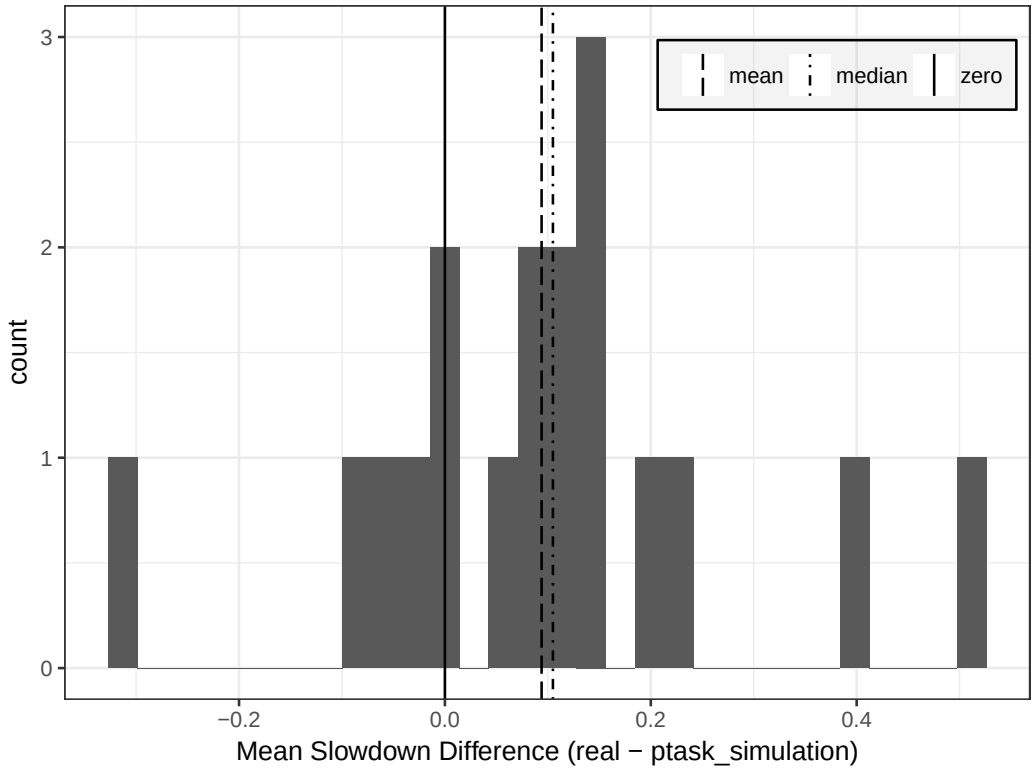


Figure 4.6: Distribution of the mean slowdown difference between real and simulated (with parallel task profiles) executions of all workloads. Ideally, the results would be centered around zero — the black vertical line. The mean and the median of the mean slowdown difference among all workloads is also represented as vertical lines.

experiment on a grid from a user-defined workflow [YB05], but it is not well suited for computer science experiments, which also need to select the underlying software stack and OS. Hopefully, computer scientists dedicated testbeds such as Grid'5000 exist and allows this management level.

Batsim evaluation experiment has been made using Execo [@execo], a tool which completely automates the experiment workflow. Execo is a tool which allows Grid'5000 users to programmatically describe their experiment workflows in order to compute them on the grid. It is a Python toolbox library that allows to run local and remote processes easily. It also provides an engine to manage the parameters sweeping and an interface to the Grid'5000 testbed, which allows to design fully automated sets of experiments.

Moreover, the scripts and tools used to generate the figures of this chapter are provided in the Git repository used for the initial Batsim article [Dut+16b]. The Gantt chart visualization and comparison tool named *Evalys* is available indepen-

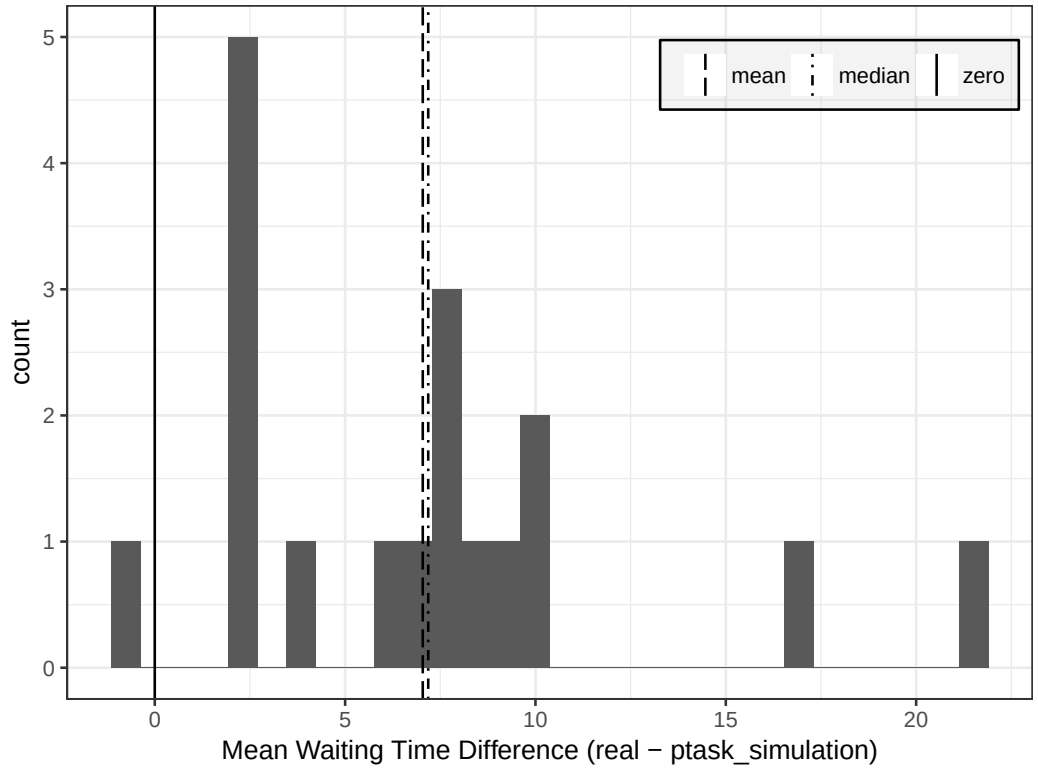


Figure 4.7: Distribution of the mean waiting difference between real and simulated (with parallel task profiles) executions of all workloads. Ideally, the results would be centered around zero — the black vertical line.

dently [@evalys]. The complete experiment workflow made to conduct this chapter experiment is also available in the initial Batsim article repository [@batexpe]. This repository also contains a Batsim workflow description, which easily allows to repeat the simulated executions and the results analysis.

Inputs and results The original input data are crucial in the process of reproducibility. Most of the inputs and results of the experiments we have done are available in the aforementioned Git repository. The results that did not fit in the repository because of their size — notably the MPI instrumentation traces ($\simeq$ 20Go) — have been shared using Academic Torrent [Mer16]. Unfortunately, this original data has been lost by lack of a persistent storage server. Similar data can be reconstructed from the information described in this section.

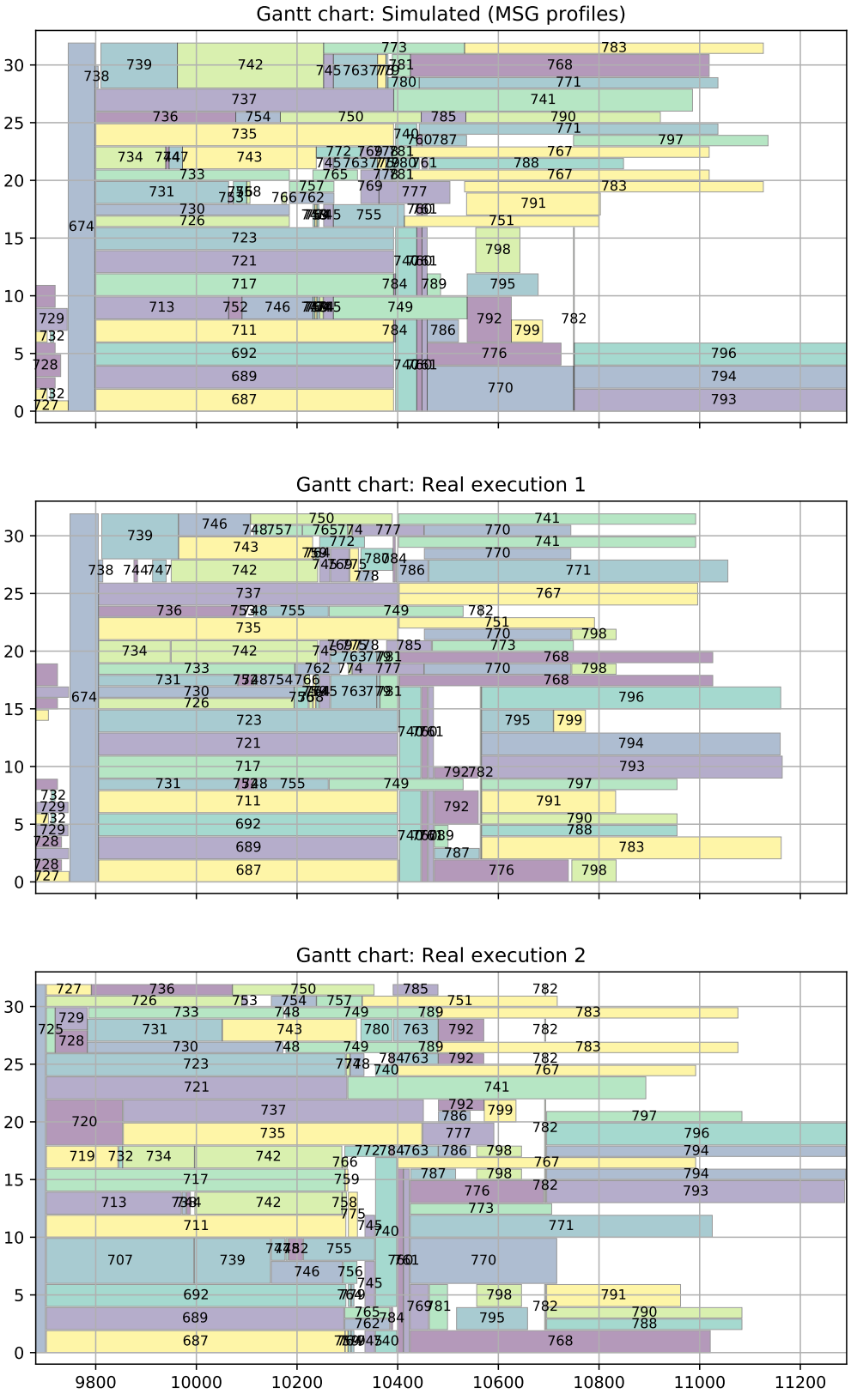


Figure 4.8: Final section of the Gantt charts of the executions of workload $seed = 6$. The uppermost Gantt chart is a simulated execution while the other two are real executions. Workload $seed = 6$ is the least stable workload in makespan.

4.10 Conclusion, Limitations and Future Work

This chapter presented Batsim, a modular RJMS simulator that provides different levels of realism. Almost any scheduling algorithms can be easily connected to Batsim, which therefore eases the comparison of existing management algorithms. Batsim input workloads are extensible and allow painless workload generation. For the sake of convenience, we provide converters from SWF workloads to Batsim ones. Batsim outputs provide clear information about scheduling metrics, job placement and energy consumption. The CSV format allows them to be easily linked with standard data analysis tools. Visualization tools are provided separately.

We used the OAR RJMS to evaluate whether Batsim matches the behavior of real RJMSs. As we want to promote experiment reproducibility, all the materials necessary to understand and reproduce our evaluation experiments are provided online. This experiment did not emphasize differences between the delay and parallel task job models, as highly homogeneous platforms have been used and no congestion has been observed during the workloads execution. As a future work, we can think of a validating process concerning the parallel task profile type, which may focus on real heterogeneous platforms.

We chose not to overfit OAR's procedures, which impacts the result realism on different metrics such as the mean waiting time. Since our architecture allows to model finely the different RJMS procedures, it would be beneficial to allow Batsim users to parametrize how the different procedures should be done in order to improve accuracy.

We are well aware that the workloads used in our evaluation process remain small in their number of resources, their number of different jobs and in their duration. We would like to do larger scale experiments but finding funding to conduct this kind of study becomes problematic, as the energy and financial costs of reservations for such experiments would skyrocket.

At the moment, Batsim allows to use a production scheduler in simulation. In this case, Batsim is in charge of simulating the RJMS whereas the decision layer of the real RJMS makes the decisions. The other way to connect the two components — plugging any Batsim-compatible algorithm to a real RJMS — has not been evaluated yet and would be an interesting future work.

The main Batsim limitation is performance. Simulators specifically optimized for simple job models — e.g., fixed-length black boxes — and algorithms — e.g., EASY backfilling [MF01] — are much faster than Batsim in these cases. Evaluating Batsim

performance and optimizing it should be an interesting future work. Batsim inner mechanics could for example be executed in a simplified and optimized version for simple simulation cases. We could also propose an auxiliary way to connect the decision-making component in addition to the current one if needed — through library calls for example.

Batsim is fully operational as I wrote these lines and its modular design makes room for many new features. We are for example interested in IO-related problems that occur when big data workloads are used. Parallel and distributed file systems simulation is being studied in Batsim but has not been validated yet. Batsim allows dynamic workload generation. We plan to improve this feature to implement more complex workload procedures — e.g., resampling with feedback [Fei16] — and thus to allow any Batsim-compatible algorithm to benefit from them.

Batsim is an open source [@batgit1] project and we encourage any researcher or engineer that has to deal with resources and jobs scheduling to use it. We would be pleased to collaborate with anyone who wants to port an existing scheduling algorithm to Batsim, or to include a Batsim-compatible algorithm support into a real RJMS.

5 Communication Models Insights Meet Simulations

5.1 Introduction

Large scale high performance computing platforms are becoming increasingly more complex. Determining efficient allocation and scheduling strategies that can adapt to their evolutions is a strategic and difficult challenge. We are interested here in the problem of scheduling jobs in hierarchical and heterogeneous large scale platforms. The application developers submit their jobs in a centralized waiting queue. The job management system aims at determining a suitable allocation for the jobs, which all compete against each other for the available computing resources. The performances are measured by some objectives like the maximum completion times or the slowdown. The most common scheduling policy is First Come First Served (FCFS), which takes the jobs one after the other according to their arrival times with backfilling (BF), which is an improvement mechanism that allows to fill idle spaces with smaller jobs while keeping the original order of FCFS.

In practice the job execution times depend on their allocation (due to communication interferences and heterogeneity in both computation and communication), while theoretical models of parallel jobs are usually considering jobs as black boxes with a fixed execution time. Existing communications models do not fully reflect the network complexity and thus, simulations are required to take into account the impact of allocations.

Our goal within this work is to test existing heuristics dealing with allocation constraints, namely contiguity and locality. Contiguity forces jobs to be allotted on resources with a contiguous index (assuming that system administrators numbered their resources by proximity), while locality is a stronger constraint imposing some knowledge of the cluster structure to use allocations restricted to clusters whenever possible.

Contributions We show in this chapter that insights gained while studying theoretical models are sometimes at odd with the practical results due to shortcomings

in the models. Moreover, this work shows that connecting existing event-based scheduling algorithms to Batsim can be done effortlessly.

More precisely, we ran wide range simulations on FCFS/BF focusing on the impact of communications under several scenarios of locality constraints. The main result is to show that taking communications into account matters, but contrary to the intuition given by theoretical models, the most constrained scenarios are the best! In other words, the constrained policies allow greater gains in performances than the potential losses due to the cost of the locality constraint.

5.2 Related Work

Modeling the modern High Performance Computing platforms is a constantly renewed challenge, as the technology evolves and quickly renders obsolete the models developed for the previous generation. While interesting and powerful, the synchronous PRAM model, delay model, LogP model and their variants (such as hierarchical delay, see [GK07] for a description of these models) are ill-suited to large scale parallelism on hierarchical and heterogeneous platforms.

More recent studies [SSE06] are still refining these models to take into account contentions accurately while remaining tractable enough to provide a useful tool for algorithm design. Even with these models, all but the simplest problems are difficult and polynomial approximations algorithms have mixed results [Sin07].

With millions of processing cores, even polynomial algorithms are impractical when every process and communication have to be individually scheduled. The model of parallel tasks simplifies this problem in a way, by bundling many threads and communications into single boxes, either rigid, rectangular or malleable (see [Leu04], chapters 25 and 26). However, these models are again ill-adapted to hierarchical and heterogeneous platforms, as the running time depends on more than simply the number of allotted resources. Furthermore, these models hardly match the reality when actual applications are used [Hun15], as some of the basic underlying assumptions on the speed-up functions (such as concavity) are not often valid in practice.

With these limitations in mind, we decided to use simulations to really take into account the communications taking place within the jobs on large scale platforms. While writing a simple and dedicated simulator is always possible, it appeared more interesting to use a detailed simulator to open our work to a larger set of platforms

and job characteristics. Among the likely candidates, SimGrid [Cas+14] fulfills all our needs. In particular, the communications can be modeled either with a TCP-flow level model as used in this chapter or at the packet level for a fine grained simulation. While simulation is not always perfect [HCS11], the results we present here are hopefully giving a better insight in the practical behavior of heuristics than the theoretical models. This work led to the development of Batsim, which is detailed in chapter 4.

A complementary approach to ours is to take into account the communications within the jobs themselves by migrating processes depending on their communication affinity [Jea+13]. This approach is rooted in the application, while we are positioning ourselves at the resource and job management system level.

Most available open-source and commercial job management systems use a heuristic approach inspired by FCFS with backfilling algorithms [MF01]. The job priority is determined according to the arrival times of the jobs. Then, BF (the backfilling mechanism) allows a job to run before another job with a highest priority only if it does not delay it. There exist several variants of this algorithm, like conservative backfilling [Lif95] and EASY backfilling [MF01]. In the former, the job allocation is completely recomputed at each new event (job arrival or job completion) while in the second, the process is purely on-line avoiding costly recomputations. More sophisticated algorithms have been proposed that consider the routing schemes of the data (like topology aware backfilling introduced in [PNM09]). In this chapter, we consider that the scheduler has a very limited knowledge of the platform, which is insufficient for topology-aware algorithms.

5.3 Problem Description

In this chapter, we are interested in the problem of scheduling a set $\mathcal{J}$ of independent and parallel jobs on a computing platform composed by a set M of computational resources (nodes or processors).

Each job $j \in \mathcal{J}$ is characterized by a rigid number q_j of required resources, a wall-time $wall_j$ (which bounds the execution time: j is killed after $wall_j$ seconds) and a release time r_j. The job execution is modeled by parallel tasks (cf. section 4.4). Parallel tasks are essentially a computation matrix $comp_j$ where each $comp_{j_k}$ represents the amount of computation on the k^{th} resource allocated to job j, and a square communication matrix $comm_j$ of size $q_j \times q_j$ in which each element $comm_j[r, c]$

represents the amount of communication from the r^{th} resource to the c^{th} resource of job j allocation.

Each resource $i \in M$ has a computing speed cs_i. The resources are connected via a network. The network links have both a bandwidth and a latency. Each resource i has a unique identification number id_i between 0 and $|M| - 1$.

Since we are interested in the online version of this problem, the scheduler only knows that job j exists once it is released. Two jobs cannot be processed at the same time on the same computational resource. Each job must be computed exactly once. The jobs cannot be preempted. The scheduling algorithms are considered as oblivious about the jobs inner settings $comp_j$ and $comm_j$. Furthermore, the algorithms know little about the platform — i.e., they only know the number of computational resources and their identification numbers.

5.4 Simulation Framework

As stated in section 5.2, we turned to simulations to evaluate many batch scheduling algorithms to check whether theoretical models match the practical experience. The added benefit over real experiments is that simulation enables reproducibility, and can easily be extended to test a very large number of parameters. The founding principle of our work is to use an existing platform simulation framework and to add a scheduling layer on the top of it. This approach allows us to take advantage of the simulation accuracy and the scalability of recognized software and allows separation of concerns since we are not simulation experts.

The survey [Cas+14] compares state-of-the-art simulators that could interest us. We chose to use SimGrid because it allows heterogeneity in both computing speed and in network links latency and bandwidth, has a good TCP flow network model, can be used easily (thanks to a good documentation and a lot of examples), is fast, has little chance of becoming unmaintained (still actively developed after 10 years of existence) and comprises features that we may use in the future — e.g., MPI applications simulation. The work conducted in this chapter led to the development of the Batsim simulator — which is described in chapter 4.

5.5 Evaluation

5.5.1 Platform and Jobs Description

Since we want to assess how the algorithms presented in [Luc+15] behave within realistic simulations, we use the same kind of platforms that the chapter described. Our platforms include sets of closely located computational resources called *clusters*. Each cluster c is a tree formed by a switch s_c and a set of computational resources all directly connected to s_c. The cluster switch s_c has an internal bandwidth bw_{s_c} and an internal latency lat_{s_c}. All resources within the cluster c have the same computing speed cs_c. All the links between s_c and the resources within the cluster c have the same bandwidth bw_c and latency lat_c. The clusters are connected together via a unique switch b whose shared bandwidth is bw_b and whose latency is lat_b. The implementation of the algorithms presented in [Luc+15] constrains all the clusters to have the same size. We chose to keep the parameters they used, which are 8 clusters of 16 computational resources each, leading to a total of 128 resources per platform.

In the following experiments, each run instance consists of a platform, a workload and a scheduling algorithm. Every generated workload consists of 300 jobs extracted from the cleaned trace (in the SWF format) of the CEA-Curie supercomputer. Our job selection criteria were to remove jobs that cannot fit entirely in one cluster and, in order to obtain interesting workloads, to ensure the resulting schedule makespan is not fixed by the longest job. Tiny jobs fit easily in the backfilling and very big ones are usually in specific queues, we then decided to only keep jobs whose execution time p_j is between two bounds $p_{lower} \leq p_j \leq p_{upper}$. Typical values for the bounds are $p_{lower} = 1$ hour and $p_{upper} = 1$ day. The method used to select the jobs is to first remove every job that does not fit our criteria then to randomly pick 300 jobs depending on a given random seed.

Since the trace only contains execution times, without any detail of actual computations or communications patterns, we chose to use basic homogeneous patterns and to create the amounts from the real jobs execution times. Let rp_j denote the real execution time of job j in the trace file. Let $rwall_j$ denote the user-given wall-time of job j. Let F_{comp}, F_{comm} and F_w respectively denote the computation factor, the communication factor and the wall-time factor. For each job j, the computation row matrix $comp_j$ is computed via $comp_j = R^1_{q_j} \times rp_j \times F_{comp}$, where $R^1_{q_j}$ is a row matrix of q_j columns of 1. For each job j, the communication square matrix $comm_j$ is obtained with the following formula: $comm_j = S^1_{q_j} \times rp_j \times F_{comm}$, where $S^1_{q_j}$ is a

square matrix of size $q_j \times q_j$ of 1. For example, $R_3^1 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ and $S_2^1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$. For each job j, q_j is read from the trace and $wall_j$ is chosen big enough to ensure the job will not be killed via the following formula $wall_j = max(rwall_j, rp_j \times F_w)$. With small wall-times, the jobs allocations would not matter since jobs would not be allowed to complete and would simply be killed after the same amount of time in any placement. Finally, the release time of each job j is set to 0 to remain in the same experimental setting as in [Luc+15], which will allow us to analyze the difference between our results and the previous ones.

5.5.2 Competing Heuristics

The scheduling algorithms compared in this chapter are variants of the well-known conservative backfilling algorithm [MF01], which targets the minimization of *makespan* (completion time of the last running job). This algorithm maintains two data structures. The first one is a list of queued jobs and the times at which they are guaranteed to start execution. The other is a profile which stores the expected future processor usage. This profile is usually a list of consecutive time slices, which stores the resources status for each time slice.

When a new job j_n is submitted, the profile is traversed in order to find a *hole* in which j_n would fit, depending on the job width $wall_{j_n}$ and height q_{j_n}. Let us suppose that the profile traversal is done by ascending date, and that this procedure returns the different holes in which j_n may fit. When a fitting hole is found, it is either accepted or rejected. If accepted, the scheduling algorithm must select the resources to allocate to j_n within the hole. Otherwise (if the hole is rejected) the profile traversal continues and future fitting holes will be found until one is accepted.

The algorithms studied in this chapter differ in their last phase, which consists in accepting or rejecting the current hole and selecting which resources to allocate to j_n in case of acceptance. A detailed description of these variants and their pseudo-code is given in [Luc+15]. In the remaining of this section, j_n will denote the newly submitted job, $H \subseteq M$ will denote the set of available resources in the current hole, and $A \subseteq H$ will denote the resources allocated by the scheduler — i.e., the selection of resources within H on which the job j_n will be executed.

The **basic** variant always accepts the first fitting hole and selects the first resources — i.e., $A \subseteq H$ such that $|A| = q_{j_n}$ and $\sum_{i \in A} id_i$ is minimal.

The **best effort contiguous** variant always accepts the first fitting hole and selects a continuous block of resources if possible. In this context, the contiguity of the set of resources A means that there exist resources with contiguous indexes. If there is no contiguous set of resources of size q_{j_n} in H, this variant selects the first resources — just as the basic variant would do.

The **best effort local** variant always accepts the first fitting hole and selects a local set of resources if possible. Otherwise, it returns the first resources as the basic variant would do. In the context of this chapter, A is said to be a local set of resources if all the resources in A are located in the same cluster.

The **contiguous** variant forces the contiguity constraint on A. Consequently, this variant rejects the first fitting holes if they do not match the contiguity constraint.

The **local** variant forces the locality constraint on A. Consequently, just as in the case of the contiguous variant, the local variant rejects the first fitting holes if they do not match the locality constraint.

Thanks to the article [Luc+15] authors, we were able to directly use their algorithms implementation in conjunction with Batsim.

5.5.3 Homogeneous Platform Experiment

The goal of the first experiment is to compare the behaviour of the different scheduling algorithms when the job amount of communication is increased on the same homogeneous platform. The jobs of this experiment were generated with 20 random seeds (0 to 19), leading to 20 different base workloads. We picked $F_{comp} = 10^6$ and $F_w = 10^3$, and 40 different values for the F_{comm} parameter have been used which correspond to a linear variation starting from 0 with steps of 10^7. In order to obtain jobs with interesting execution time — i.e., to avoid that the resulting schedule makespan is only fixed by the longest job — we set $p_{lower} = 1\ hour$ and $p_{upper} = 4\ hours$. All the clusters of the platform used in this experiment are the same and defined by the following parameters. $bw_{s_c} = 1.25 \cdot 10^9$, $lat_{s_c} = 0$, $bw_c = 1.25 \cdot 10^6$, $lat_c = 24 \cdot 10^{-9}$. The platform main switch parameters are $bw_b = 1.25 \cdot 10^9$ and $lat_b = 24 \cdot 10^{-9}$. This platform is derived from the existing Grid'5000 Griffon cluster whose platform description was available in the SimGrid examples. The combination of these parameters created 4000 instances (800 per scheduling algorithm variant).

Figure 5.1 shows the makespan of the resulting schedule of every run instance of the first experiment. Additionally, a linear trend line has been computed for a better

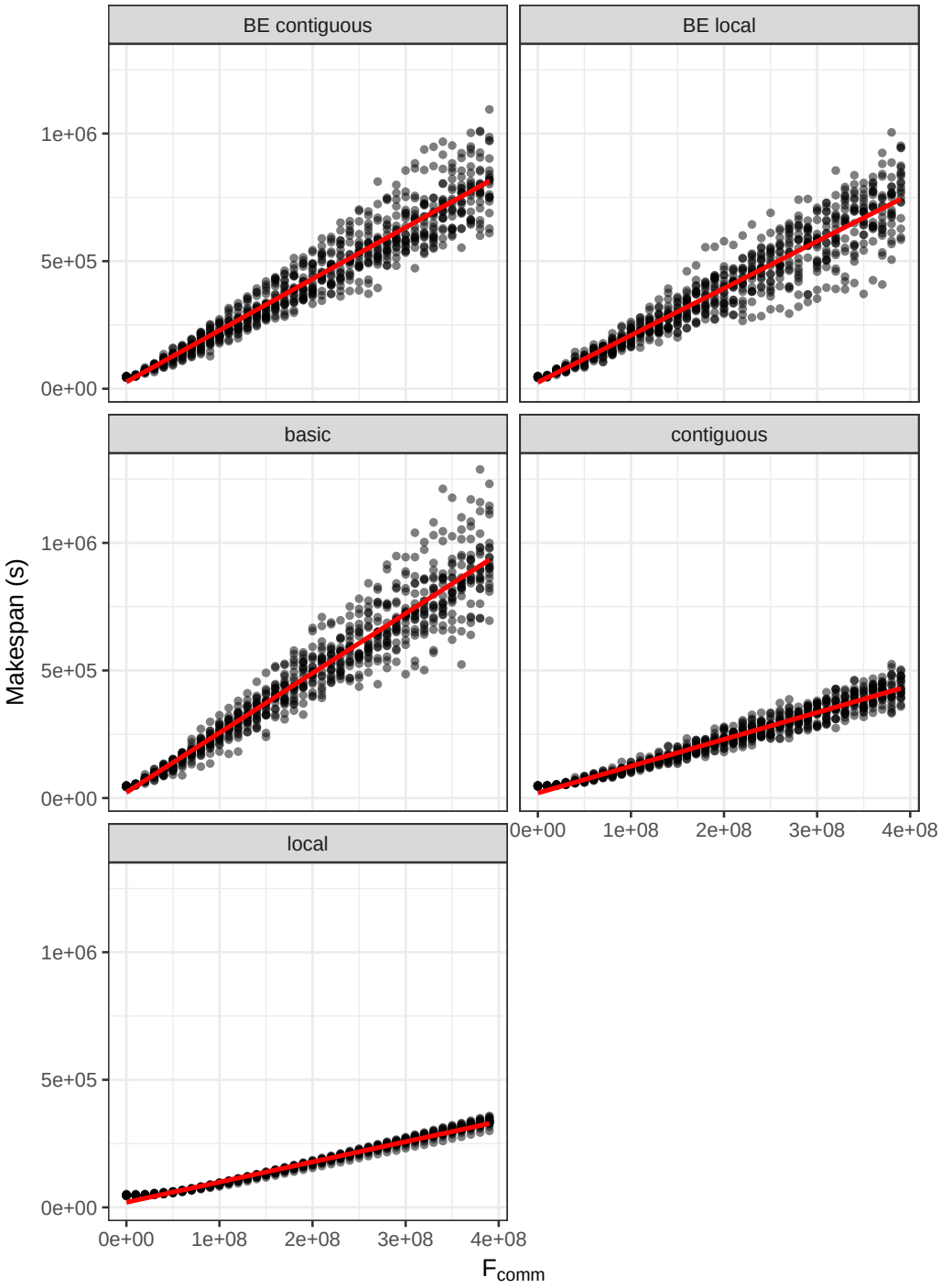


Figure 5.1: The makespan of every run instance in function of the communication factor F_{comm} for the homogeneous platforms experiment. Each facet corresponds a scheduling algorithm. Each point corresponds to a schedule (800 points per scheduling algorithm).

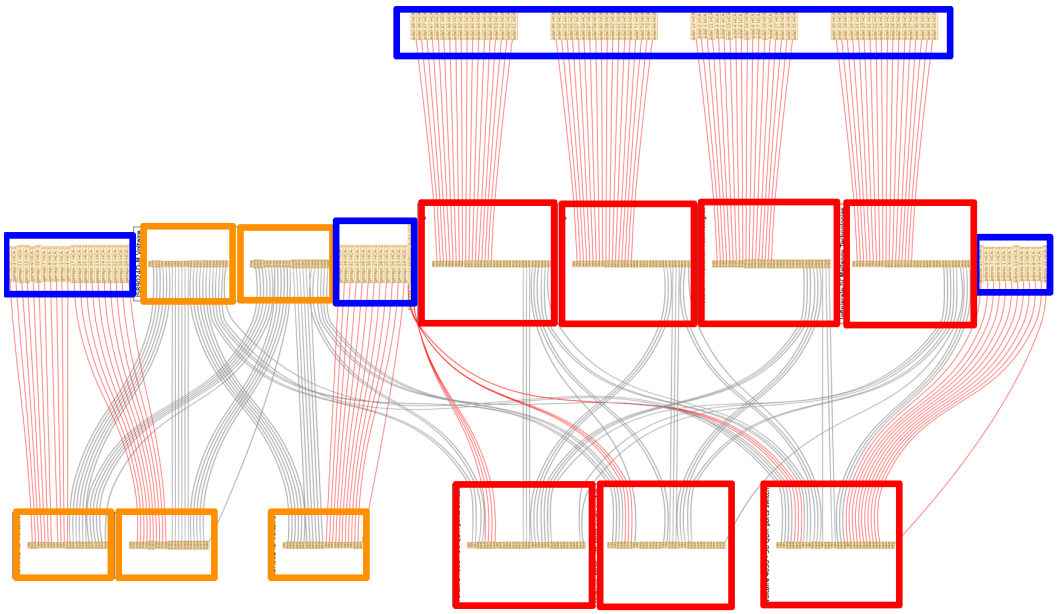


Figure 5.2: Grid'5000 cluster architecture in Grenoble. The red rectangles are 40 GB/s InfiniBand switches, orange rectangles are 20 GB/s InfiniBand switches, while the blue rectangles are three different families of computing nodes.

comparison of the heuristics. The basic algorithm (as defined in the previous section) is completely without constraints and has the worst performance of all competing heuristics. Imposing contiguity without any knowledge of the underlying structure gives better performances than basic, while knowledge of locality further improves the results. More surprisingly, the strict heuristics are outperforming the more relaxed heuristics, even though strict heuristics delay some jobs if the constraints cannot be matched. Furthermore, the makespan induced by the forced constraints are much more stable than their best-effort counterparts.

5.5.4 Heterogeneous Platform Experiments

In addition to the homogeneous platform experiment, we conducted two experiments on heterogeneous platforms. The goal of these two experiments is the same as in the homogeneous case: Assessing how the algorithms behave when the amount of communication within jobs is increased. However, these experiments focus on many heterogeneous platforms instead of one homogeneous platform, to more closely reflect the existing clusters in our computing centers. For example, Figure 5.2 gives an idea of the layout of the Grid'5000 cluster in Grenoble[1].

[1]For more details, a larger version of the figure is available at:
`https://www.grid5000.fr/mediawiki/index.php/Grenoble:Network`

Table 5.1: The parameters of the clusters used in heterogeneous experiments. These values are multiplication factors of our base cluster b whose values are $bw_{s_b} = 10\ Gbits \cdot s^{-1}$, $lat_{s_b} = 0\ s$, $cs_b = 286.097 \cdot 10^3\ flop \cdot s^{-1}$, $bw_b = 10\ Gbits \cdot s^{-1}$, $lat_b = 24 \cdot 10^{-9}\ s$.

c	bw_{s_c}	lat_{s_c}	cs_c	bw_c	lat_c
c_1	2	0	1	1	1
c_2	4	0	2.02	1	1
c_3	1	0	2.94	1	1

c	bw_{s_c}	lat_{s_c}	cs_c	bw_c	lat_c
c_4	1	0	1.24	1	1
c_5	2	0	1.61	1	1
c_6	1	0	1.72	1	1

In order to remain realistic in the kind of platform heterogeneity to simulate, we analyzed the network of several Grid'5000 sites and ran a linear algebra benchmarking tool on many machines to have an idea of how much the node computing speed may vary within one site. Our results on the Rennes and Grenoble site showed that the network bandwidth might vary between 1 and 4 and that the node computing speed may vary between 1 and 3. More precisely, with our benchmark the computing speed in the Rennes site were 1, 2.02 and 2.94 times more powerful than the lowest one. On Grenoble we obtained computing speeds of 1.24, 1.61 and 1.72 times the lowest one. We therefore decided to create a set of lowly heterogeneous platforms and a set of highly heterogeneous platforms and see how the different scheduling variants behave on such platforms.

The two heterogeneous experiments use six clusters whose parameters can be found in Table 5.1. The first heterogeneous experiment uses four platforms formed by 3 clusters c_1, 3 clusters c_2 and 2 clusters c_3. The four platforms differ by the ordering in which the clusters are in the platform. The used orderings are by ascending computing speed $o_1 = (c_1, c_1, c_1, c_2, c_2, c_2, c_3, c_3)$, by descending computational power $o_2 = (c_3, c_3, c_3, c_2, c_2, c_2, c_1, c_1)$ and other orderings $o_3 = (c_1, c_2, c_2, c_3, c_3, c_2, c_1, c_1)$ and $o_4 = (c_3, c_1, c_2, c_3, c_1, c_2, c_1, c_2)$.

The workloads of this experiment have been generated with 10 random seeds (0 to 9). We used $F_{comp} = 10^6$, $F_w = 10^3$, and 20 different values for the F_{comm} parameter corresponding to a linear variation starting from 0 with steps of $2 \cdot 10^7$. The processing time bounds to pick the jobs were $p_{lower} = 1\ hour$ and $p_{upper} = 4\ hours$.

The second heterogeneous experiment is exactly the same as the first but its platforms use clusters c_4, c_5, c_6 instead respectively of clusters c_1, c_2 and c_3. We call the first experiment highly heterogeneous because the resource computational power varies from 1 to 3 and the network bandwidth from 1 to 4 within it. We call the second experiment lowly heterogeneous because these amounts does not vary as much as in the first experiment. Each experiment consists of 4000 run instances (800 per scheduling algorithm variant).

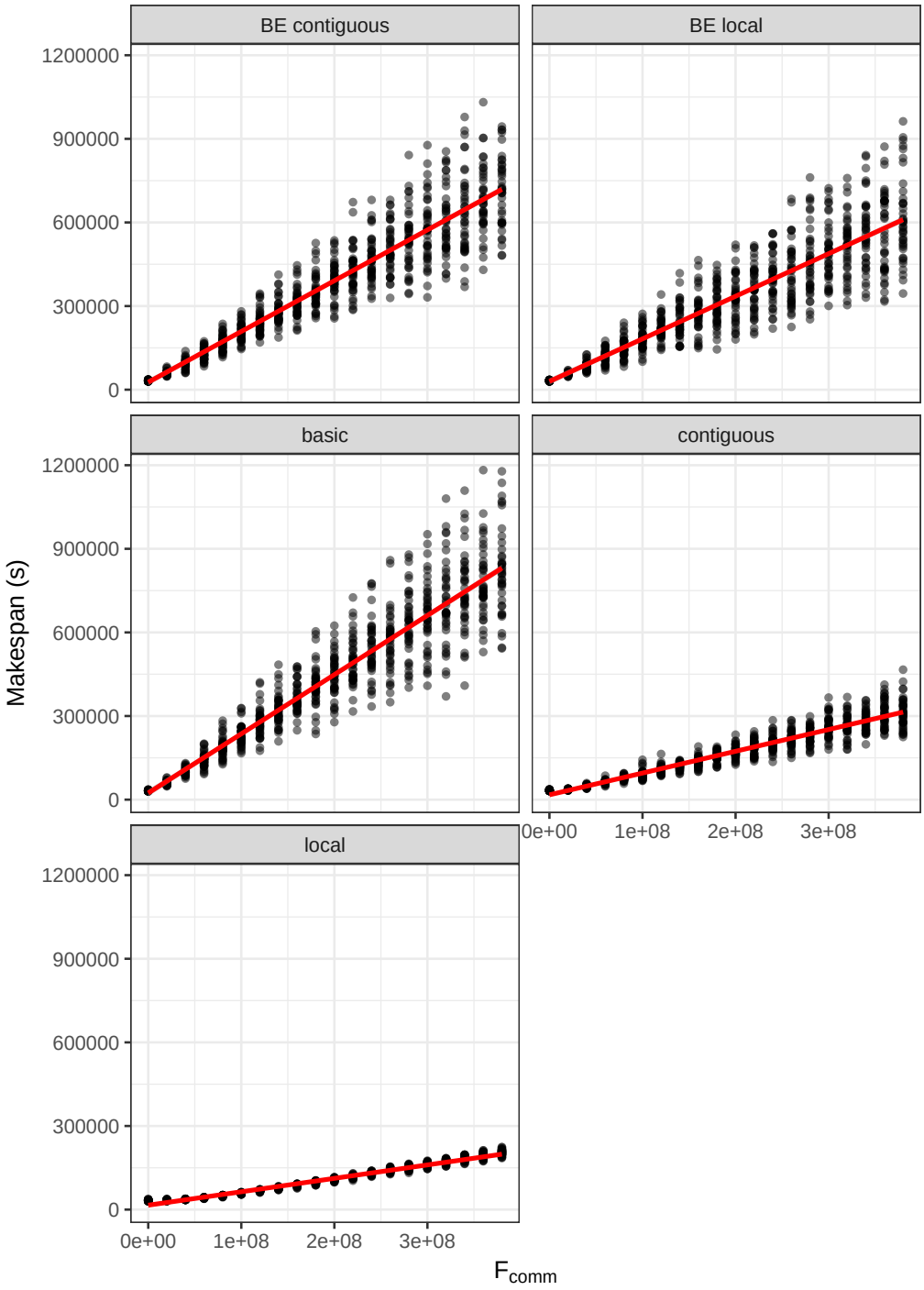


Figure 5.3: The makespan of every run instance in function of the communication factor F_{comm} for the heterogeneous platforms experiment. To each facet corresponds a scheduling algorithm. Each point corresponds to a schedule (1600 per scheduling algorithm).

Figure 5.3 shows the makespan of the different scheduling algorithm variants when the amount of communication is increased in the heterogeneous experiments. These graphs do not differ greatly from the homogeneous case. For the makespan, the forced constraint variants scale better and are more stable than their best-effort counterparts when the amount of communication within jobs is increased. Furthermore, we did not notice any impact of the cluster ordering within one platform on the resulting schedules makespan. We did not notice a great difference between the slightly heterogeneous platforms and the highly heterogeneous ones neither, that is why the results of the two experiments have been plotted together. The most notable result is that in a heterogeneous setting, the locality knowledge is much more important as the gap between the basic heuristic and the locality aware is greatly increased.

5.6 Conclusion

The purpose of this work was to show through simulations if theoretical models are giving pertinent insight on job scheduling on large scale hierarchical and heterogeneous platforms. The main hypothesis we tested was that enforcing contiguity or locality would not degrade the performance. The results clearly show that the constraints are beneficial to the schedules, by reducing the communication times. More broadly, this shows that models where internal communications are hidden within parallel tasks are very ill-suited to current architectures, and should be reevaluated. This work also showed that connecting existing event-based scheduling algorithms to Batsim can be done easily, which allows the exploration of wider scenarios.

6 Energy Budget Control in HPC With Energy-Aware Resource Management and Job Scheduling

6.1 Introduction

Modern supercomputers run on huge amounts of electrical power. For instance Sunway TaihuLight — the leading system of the TOP500 — develops 93 petaflop/s and consumes a power of 15 megawatts [@sunwaytl]. The electricity bill of such platforms can roughly equal their hardware cost if we consider their entire lifespan. Energy consumption is the most important limit to build exascale machines [Don+11].

Multiple techniques have been developed to control the energy consumption of such huge platforms. In particular, power capping limits the instantaneous power consumption to a certain threshold. Limiting the *power* during a time period leads to controlling the *energy* consumption — as the energy is the integral of the power over time. The survey provided in [Bat+15] gives a thorough analysis of related work on power management strategies along with details upon the relationship between supercomputing centers and electricity service providers in the US. Among the different studied techniques, power capping in conjunction with job scheduling and shutdown mechanisms appeared as the most promising. Employing these methods allows to manage energy consumption through the control of the instant power consumption. Such methods nevertheless lack in adaptability, as power is controlled independently of the instant load.

A recent study [Pat+16a] implicating a larger group of supercomputers and electricity providers — in both the US and Europe — showed that while the upper power bound is an important parameter, power variations do not affect the final energy cost in most use cases. In this chapter, we show that adopting flexible power adaptive scheduling techniques by setting restriction on *energy* consumption instead of power

can optimize system utilization, slowdown and even energy efficiency in comparison with a rigid power capping strategy.

We present an adaptation of a standard job scheduling algorithm that is able to limit the energy consumption during a time period. It is similar to power capping, with the difference that instead of limiting execution under a maximum instantaneous power consumption, the limit is set on the maximum energy that can be consumed during a particular time period. The developed techniques are extensions to the backfilling mechanism [MF01]. Instead of considering only the availability of computing resources, they also take the availability of energy into account. Overall, they enable the platform to meet a certain energy consumption budget. The reduction of energy consumption is achieved by idling the nodes or by shutting them down opportunistically. Experiments through intensive simulations show that our techniques keep high performances while respecting specific energy budget objectives.

The problem studied in this chapter is detailed in section 6.2. Existing approaches to control power and energy are then presented in section 6.3. Section 6.4 presents our new algorithms to support energy budgeting. Section 6.5 evaluates how the proposed algorithms behave with actual log data in simulation. Finally, conclusions and future works are presented in section 6.6.

6.2 Problem Description

6.2.1 Scheduling Jobs in HPC

A job is an application that a user wants to execute on a computing platform. The scheduler relies on a scheduling algorithm to determine when and where the jobs should be executed. Only limited information is known about the jobs in practice. Users typically specify the number of computational resources they need and an execution time upper bound, which is called the wall-time — jobs that reach their wall-time are killed by the system. In this work, we define a job j by its release time r_j (the time at which the job is submitted by the user), its number of requested processors q_j, an estimation of the running time $wall_j$. The job execution time is denoted by p_j and is not known in advance but only when j completes.

The network hierarchy existing in most HPC centers is not explicitly considered in this chapter. More simply, we consider that all processors are totally ordered and that a job must run on neighboring processors. This simplification supposes that the resources are numbered by proximity and that users want their jobs to be executed

on resources close to each other — as it is likely to to reduce the communication times and therefore the jobs duration [Luc+15].

6.2.2 Energy and Power-saving Techniques

Various techniques can be used to save energy on different levels. Some of these techniques are introduced by architecture manufacturers — e.g., DVFS — while others are invariant possibilities of the infrastructure — e.g., node on/off or heterogeneity. In this work, we only take shutdown techniques into consideration and consider that processors can be switched-off independently.

We are interested in homogeneous machines that have various states $S = \{ computing, idle, off, on \rightarrow off, off \rightarrow on \}$. At a given time t, machine i is in one and only one of the states of S. Idle machines can be switched-off, which takes a time $t_{on \rightarrow off}$. Off machines can be switched-on, which takes a time $t_{off \rightarrow on}$. The power consumption of machine i is fully determined by its state. A fixed power consumption is associated to each state. Explicitly, the power consumption of states are denoted by $P_{computing}$, P_{idle}, P_{off}, $P_{on \rightarrow off}$ and $P_{off \rightarrow on}$.

Despite the simplicity of this model, we think that it meets our needs for two reasons. First, the experiments that we have conducted (see section 6.5.1) show that it seems sufficient for our use case. Second, more precise measures of energy would have a prohibitive cost. A hardware cost as integrated energy sensors are not accurate enough [Geo+14], but also a software architecture cost — dedicated software for accurate energy measurements should be added over the cluster — and a data management cost to store and analyze the data. Using basic energy measurement reduces the cost of all these steps.

Readers may wonder why we do not take DVFS into account, as it can be used to control the power consumption of a job. Previous studies [GGT15; Geo+15] have shown that controlling the energy consumption of jobs with DVFS is not trivial. Depending on the type of application, a given DVFS value may either increase or decrease the total energy consumption of the application. Thus, without a precise knowledge about each job, the scheduler cannot guarantee that a given DVFS value will decrease the energy consumption. Since the scheduler usually does not have this type of information, we think that dynamically adapting the DVFS to reduce the energy consumption should be done within the job itself and **not** at the platform scheduling level, as it will result in better energy efficiency. Our approach is totally compatible with job-level DVFS optimizations.

Specifically, we are interested in the problem of scheduling jobs on a large number of resources with the following constraint. During a certain time frame — starting at t_s and ending at t_e — the energy consumption of the whole computing platform cannot exceed a given limit B. The energy limit we are using here (in joules) must be distinguished from an electrical power limit (in watts). Budget periods can be successive but we will study only one time frame for the sake of clarity and conciseness.

6.2.3 Scheduling Algorithms Evaluation

No perfect scheduling objective exists [FF05]. We will consider three different metrics in this chapter. The first metrics is the utilization — i.e., the proportion of processors that are used during a time period. This objective is mostly used by cluster owners, as it may represent the cluster productivity.

The bounded slowdown [Fei01] is more end-user centered. It measures the satisfaction of end-users. Most of the time its average is computed as defined in equation 6.1, where $wait_j = start_j - r_j$ is the waiting time of job j ($start_j$ is the moment at which job j starts to be executed) and τ is the bounding constant (generally set to 10 seconds in the literature).

$$\text{AVEbsld} = \frac{1}{n}\sum_j \max\left(\frac{wait_j + p_j}{\max(p_j, \tau)}, 1\right) \tag{6.1}$$

The third metrics is the amount of energy consumed. In this chapter we do not try to reduce the energy consumption but to control it during a period of time.

6.3 Related Work

6.3.1 Controlling Power and Energy Consumption

Many papers focus on controlling the power consumption [Eti+12a; Rou+12; Sar+14]. In these studies, the objective is to control the final energy cost of the cluster while keeping good performances. PATKI et al. [Pat+15] argue that thanks to the control of power consumption, one can buy a bigger cluster for the same annual price. A bigger cluster improves the allocations and the scheduling performances.

Power cap mechanisms have the two major drawbacks of requiring high knowledge about the applications — to tune DVFS or a similar technique — and of delaying some jobs. In our previous study [GGT15] we found that only controlling the power increases the turnaround time of big jobs — as it is harder for them to *fit* in the power cap. This is one of the reasons that led us to focus on energy budgeting, as we want to keep the benefit of controlling the cost of the cluster without discriminating against any type of job.

Energy budgeting has been studied for a long time in embedded systems [Bam+16] as these systems are mainly limited by their battery capacity. Nevertheless, we cannot use the results from this field because they are applied to real-time small-scale systems.

In [MV15], MURALI et al. study a metascheduler that controls multiple HPC centers. The objective is to reduce the overall cost by adapting the energy consumption to the electricity price of each different cluster. YANG et al. [Yan+13] consider a scheduling problem with two periods. One during which an energy limit is set, and the other one without energy limit. While this approach is interesting, the proposed algorithm is not scalable and is hardly extendable with other constraints. In the study [Khe+14] KHEMKA et al. maximize a *utility* function in a cluster with daily energy budget. They solve the problem thanks to an offline heuristic. Instead of relying on an utility function, we use classical scheduling objectives as described in section 6.2.3.

The energy consumption of a shutdown node is very small [BH07]. The technique called opportunistic shutdown takes advantage of this power saving. This technique consists of shutting down the nodes that are idle — nodes are monitored to this end. As soon as a defined idle period is witnessed, the decision of shutting them down is taken. As shown in [HHN08], such a solution could lead to non-negligible energy savings. This solution has however some limitations. One of them is the cost — in both time and energy — involved by switching nodes on or off. Going off and on again can take several minutes at maximum power [OLG08].

6.3.2 Resource and Job Management Systems

Current high-performance computing centers contain thousands of computing nodes, which can amount to millions of cores. These computational resources are managed by one software called the Resources and Jobs Management System (RJMS). This software is in charge of monitoring the resources and of executing parallel jobs on them.

Managing resources at this scale compels the scheduling algorithm to be very efficient. Therefore, greedy algorithms such as EASY backfilling [MF01] are commonly used in HPC centers. Unfortunately, these algorithms do not take energy consumption into account during their decision process.

Algorithm 1: EASY backfilling algorithm

```
for job ∈ queue do
    if system has enough processors to start job now then
        launch job;
        remove job from queue;
    else
        break;
    end
end
firstJob = pop first element of queue;
Reserve processors in the future for firstJob;
for job ∈ queue do
    if system has enough processors to start job now and does not overlap with
     firstJob reservation then
        launch job;
        remove job from queue;
    end
end
Remove the processor reservation of firstJob;
Push back firstJob at the top of queue;
```

The EASY backfilling algorithm — summarized in Algorithm 1 — is one of the most widely used scheduling algorithms in the systems we are interested in. This algorithm only focuses on the present time since the future is unpredictable[1]. The EASY backfilling policy is rather aggressive, as all jobs but the first one in the queue can be delayed by backfilled jobs. This behavior tends to increase the resource utilization rate. The popularity of this algorithm can then be explained by: 1. the ease of implementation, 2. the ease of extending the basic policy, 3. the high resource utilization rate implied by this aggressive backfilling policy, and 4. the scalability of being present-focused.

[1] In the problem studied in this chapter, most events cannot be known beforehand — such as node failures, jobs submissions and completions.

6.4 Proposed Algorithm

6.4.1 Desired Properties

The proposed algorithm should obviously comply with an energy budget during a given period. This energy budget should be strictly respected. Moreover, we want the algorithm to be modular enough to support extra features such as opportunistic shutdown. As we target large scale platforms, we want the algorithm to be efficient and scalable. Finally we want the algorithm to avoid dramatic changes over currently implemented solutions — for the purpose of maximizing its adoption.

6.4.2 Algorithm Description

As EASY backfilling — summarized in Algorithm 1 — has all the desired properties but the energy ones, we chose to base our algorithm on it. We defined two rules that our algorithm must respect under all circumstances to comply with the energy budget.

- Rule 1: Avoid spending the whole budget too early, as it would unbalance the performances during the budget period. To this end, the budget's energy is made available gradually over time, at a rate of $B/(t_e - t_s)$ joules per second.
- Rule 2: Never have energy debts. Thus, before taking the decision of running a job, we have to ensure that enough energy is available for the entire duration of the job execution. This comprises taking the past, present and future power consumption of the whole cluster into account.

If we replace *energy* by *money* and *running jobs* by *buying stuff*, these two rules could describe how someone that never wants to be in debt would manage its monthly paycheck. Figure 6.1 shows the global idea behind these two rules. Energy is first saved by consuming less power than the power rate. The saved energy can then be used to execute jobs that require more energy than the power rate to be executed.

These two rules are integrated within the EASY backfilling algorithm in the following manner. A counter named C_{ea} stores the amount of available energy — i.e., the amount of energy that the algorithm is allowed to spend at the present time. C_{ea} equals to the amount of energy made available since the beginning of the budget period (via rule 1), minus the energy which has been consumed by the cluster. Whenever the algorithm checks whether enough processors are available to run a

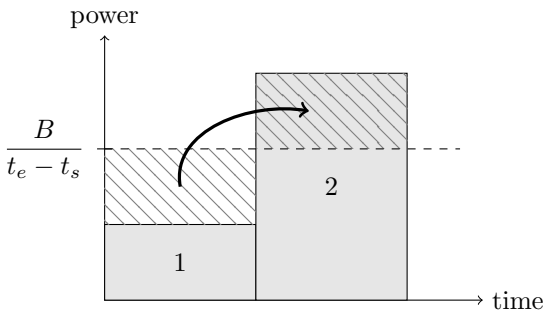


Figure 6.1: The main idea behind the proposed algorithm.
A power limit is set, but consuming less power than it results in temporary energy savings. Such saved energy can then be used to execute jobs that exceed the power limit. In this example, the energy saved during job 1 execution is used to execute job 2.

job, it also checks whether enough energy is available. The energy balance should always be positive during the entire execution time of the job — rule 2 would not be fulfilled otherwise. The algorithm uses $wall_j$ to estimate the length of job j.

Whenever the job at the top of the queue cannot be started immediately, a processor reservation is made for it — as in regular EASY backfilling. The backfilling rule dictates that other jobs might be executed before the first job if they do not delay it. Jobs cannot however be backfilled the usual fashion, as they may delay the first job by *stealing* its energy and thus break our set of rules. Consequently, our algorithm also makes energy reservation for the first job when it cannot be started immediately. The proposed solution is presented in algorithm 2. The additions that have been done over EASY backfilling are underlined and colored in brown.

6.4.3 Implementation details

Energy Consumption Monitoring

The counter C_{ea} stores the amount of available energy. It is updated whenever the algorithm is called and also at every *monitoring stage*.

Whenever the algorithm is called, C_{ea} is incremented by a certain amount by applying rule 1. Furthermore, C_{ea} is decremented depending on the cluster's energy consumption since the last algorithm call. This energy consumption is coarsely overestimated within the algorithm, by counting the number of computing and non-computing nodes. C_{ea} may also be decremented during the algorithm execution when a reservation is done.

Algorithm 2: Energy budget backfilling algorithm

```
for job ∈ queue do
    if system has enough processors
     and enough energy is available to start job now then
        launch job;
        remove job from queue;
    else
        break;
    end
end
firstJob = pop first element of queue;
Reserve processors in the future for firstJob;
Reserve energy in the future for firstJob;
for job ∈ queue do
    if system has enough processors and enough energy is available to start job
     now and does not overlap with firstJob reservation then
        launch job;
        remove job from queue;
    end
end
Remove the processor reservation of firstJob;
Remove the energy reservation of firstJob;
Push back firstJob at the top of queue;
```

Monitoring stages allow to obtain the amount of energy which has really been consumed by the cluster. C_{ea} is therefore incremented during these stages, as the algorithm always overestimates this amount of energy. The stages occur periodically. The period value must be set considering a trade-off between precision and the overhead of gathering energetic data. The more precise the monitoring of the energy is, the more precisely our algorithm will respect the energy budget. Increasing the monitoring period reduces the global costs, as 1. the algorithm is called less frequently 2. the nodes and the network are less frequently used to gather and transmit monitoring data.

Energy Consumption Estimation

The above explanation of the algorithm mentions that the algorithm must determine whether the energy balance would be positive in the future. As accurately predicting the energy consumption of a job — or a cluster — is difficult, we want our algorithm to work even if those estimations lack precision. We assume that using overestimated energy consumption values — for the jobs and the cluster — is enough to make our algorithm work.

The estimation of the power consumption of one computing processor is denoted by $\tilde{P}_{computing}$ and expressed in watts. The estimation of the power consumption of one idle processor is denoted by $\tilde{P}_{idle}$ and expressed in watts. The algorithm will overestimate the energy consumption of the cluster if $\tilde{P}_{computing}$ and $\tilde{P}_{idle}$ are overestimated. At every monitoring stage the real energy consumption is measured and C_{ea} is updated accordingly such that the overestimation is balanced. Overestimating the power consumption thus saves more energy in the counter, which will allow more jobs to be started later on.

We recommend to benchmark the cluster power consumption in certain scenarios and to use the maximum observed value to determine the aforementioned overestimations. Executing a CPU-intensive application — e.g., one from the LINPACK suite — is recommended to estimate $\tilde{P}_{computing}$, while observing the processors doing nothing may be enough to estimate $\tilde{P}_{idle}$.

Interactions with opportunistic shutdown

The proposed algorithm can be used without any modification to work with opportunistic shutdown, as off nodes consume less power than an idle ones. This leads

our algorithm to overestimate even more the cluster energy consumption, which makes greater amounts of energy available after monitoring stages.

Differences with EASY backfilling

The EASY backfilling algorithm is called whenever a job arrives or whenever some resources are made available. Our algorithm is executed at the same events but also when more energy is made available — namely at every $monitoring\ stage$. The overhead is minimal, as the full algorithm is only run if the amount of available energy is sufficient to run the first job of the queue.

When the energy budget is unlimited ($B = \infty$) our algorithm produces the same schedules as EASY backfilling. When the energy budget is very small our algorithm will start the jobs in the order of the queue.

6.4.4 An alternative similar to power capping

The proposed algorithm is close to a power capping mechanism. Making $B/(t_e - t_s)$ joules available each second is close to having a power cap limit of $B/(t_e - t_s)$. The rules introduced previously can be seen as rules that allow to violate the power cap in some cases — these cases being mostly *when energy is available*.

As power capping is widely studied and already implemented in several RJMSs, we propose a slightly modified version of our energy budget algorithm that is even closer to a power capping mechanism. In the remainder of the chapter, the already presented algorithm will be called $energyBud$, while the algorithm closer to power capping will be called $reducePC$.

The difference between $energyBud$ and $reducePC$ lies in how the jobs respect their energy reservation. In $energyBud$, the reserved energy is subtracted from C_{ea}. In $reducePC$, the number of joules made available per second is reduced — as if the power cap had been reduced. If job j makes a reservation of E_j joules at time $\tilde{start}_j$ — and thus guarantees to start at $\tilde{start}_j$ — the number of joules available per second is reduced by $E_j/(\tilde{start}_j - now)$ during the time period between now and $\tilde{start}_j$.

The main difference between the two algorithms can be observed when a short job that uses all the available processors is being backfilled while there is an ongoing energy reservation. In $energBud$ the job can be launched if enough energy is available. However the job cannot be started at the present time in $reducePC$ as the number of joules available per second has been reduced.

Measure	Value
P_{idle}	95.00 W
$P_{computing}$	190.74 W
$P_{off \to on}$	125.17 W
$t_{off \to on}$	151.52 s
$P_{on \to off}$	101.00 W
$t_{on \to off}$	6.10 s
P_{off}	9.75 W
$\tilde{P}_{idle}$	100.00 W
$\tilde{P}_{computing}$	203.12 W
$monitoring\ period$	10 min

Table 6.1: The values used to calibrate the simulator and to parametrize our algorithms.

6.5 Evaluation

The aim of the evaluation is to answer the following questions:

- How better is the proposed algorithm compared to a standard power capping mechanism?
- What is the gain of activating opportunistic shutdown?
- If the budget is reduced by 80%, are the performances also reduced by 80%?
- Which is the best between $reducePC$ and $energyBud$?

All material to produce, analyze and visualize the results of our evaluation process are available online on the following git repository [@reprEB17].

The different heuristics are evaluated with the Batsim simulator. The heuristics and mechanisms described in this chapter have been integrated into the pybatsim project [@pybatsim] — a collection of Batsim-compatible scheduling algorithms implemented Python.

6.5.1 Simulation Calibration

We have made various measurements on the Taurus Grid'5000 [Bal+12] cluster to calibrate the simulation. This cluster is composed of 16 Dell PowerEdge R720 nodes,

each with two Intel Xeon E5-2630. The nodes are equipped with wattmeters, which allows to measure precisely their energy consumption — one value every second.

In order to obtain idle-related measurements, we reserved the nodes and left them in an idle state for 200 seconds. The wattmeters generated series of power consumption values for each node, whose average over time has been computed for each node. We then computed the average and the maximum of these values over nodes to respectively obtain P_{idle} and $\tilde{P}_{idle}$. For the sake of simplicity we attribute the *per node* measurements, calculated during the calibration, to *per processor* values in our model.

We did roughly the same to obtain computation-related measurements. We just run a LINPACK benchmark on the nodes instead of letting them idle. This allowed to obtain values for $P_{computing}$ and $\tilde{P}_{computing}$.

In order to obtain switch-related measurements, we made 50 switch-on and 50 switch-off operations on each node. We considered a node as off when its power consumption reached its minimum. We considered it as on once capable of starting a new job — i.e., when all services are running and operational. This allowed us to measure the amount of time and the amount of consumed energy of each switch operation. We then chose to average these amounts to obtain $P_{off \to on}$, $t_{off \to on}$, $P_{on \to off}$ and $t_{on \to off}$.

We finally chose a $monitoring\ period$ of 10 minutes as it appears to be a good trade-off between precision and monitoring overhead. This choice is complex as it depends on the available energy sensors and the way to gather data from the computing nodes to the controlling node. This 10-minute value seems close to what is commonly used in supercomputers.

6.5.2 Testset

We chose to replay 1-week-long extracts of real traces coming from the Parallel Workload Archive [FTK14] to assess our algorithms. We chose to use 3 different traces and to extract 10 disjoint weeks from each one of them, thus leading to 30 different input workloads for our simulator. Since scheduling decisions have more impact when the utilization is high, the weeks have been selected with this criterion in mind. The original traces are:

- Curie (80640 processors, dates from 2012 and lasts 3 months),
- MetaCentrum (3356 processors, dates from 2013 and lasts 6 months),

- SDSC-Blue (1152 processors, dates from 2003 and lasts 32 months).

Every input trace is executed for its full length — 1 week in this case. However, an energy-budgeted period is applied for three days in the middle of each trace. We hope that this choice allows to observe the impact of the energy-budgeted period on the metrics during the period but also after it. In the remainder of the chapter, the energy budget is expressed as a percentage. 100% corresponds to the energy that the cluster would have consumed if all the processors on the cluster were computing during the three days. We run each input trace with the following budget values: 100%, 90%, 80%, 70%, 60%, 50%, 49% and 30%. 49% corresponds to the amount of energy that the cluster would have consumed if all the processors were idle for three days.

We evaluated 4 different algorithms. The first one is standard EASY backfilling. As this algorithm does not support energy budget, it is only executed with a 100% budget. The second one is a power capped EASY backfilling. A power limit is set during the whole energy budget period, which is set to the energy budget (J) divided by the period length (s). The platform energy consumption is estimated with $\tilde{P}_{platform} = n_{idle} \times \tilde{P}_{idle} + n_{computing} \times \tilde{P}_{computing}$, where n_{idle} is the number of idle nodes and $n_{computing}$ is the number of nodes which are computing jobs. This algorithm is roughly the same as EASY backfilling, but jobs are not executed if they cause $\tilde{P}_{platform}$ to be greater than the power limit. The last two algorithms are the ones presented in section 6.4 — namely $energyBud$ and $reducePC$.

Each algorithm that supports power budgeting is executed with and without opportunistic shutdown. When the opportunistic shutdown mechanism is enabled, idle nodes are switched-off as soon as they become idle. All the 1470 configurations comprised in our evaluation process have been executed in simulation.

6.5.3 Results

Traces do not come from the same cluster and do not have the same jobs. The different traces thus do not present the same opportunities for the algorithms to improve results. As a consequence, all measures are normalized to reduce the effect of each trace so that the results are comparable. The method described in [Mor08] normalizes the data to remove the between-subject variability. The following graphs — namely figure 6.2, figure 6.3, figure 6.4 and figure 6.5 — present the average of the normalized objectives, considering the traces as the between-subject variable.

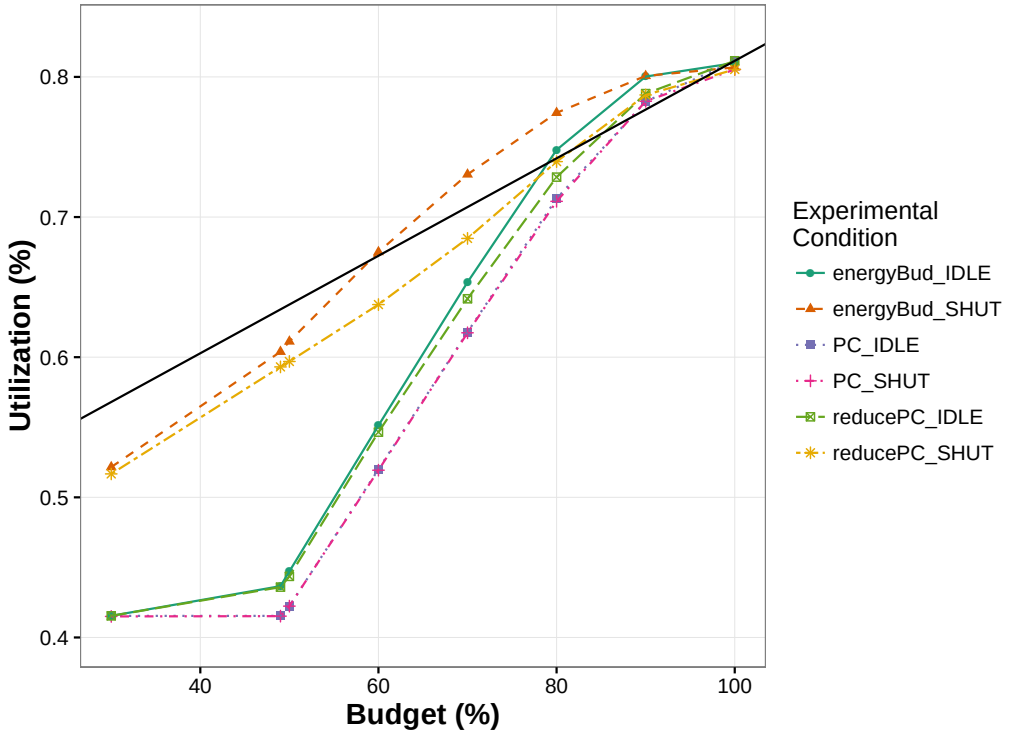


Figure 6.2: Normalized mean system utilization during the week. The black line represents a theoretical performance baseline — as detailed in section 6.5.6.

As stated in section 6.2.3, some objectives are defined for a time period while others are defined for a number of jobs. For the ones depending on a time period — namely the utilization and the relative energy consumption — we used the whole week as the time period. The jobs that have been scheduled after the end of the week are thus not taken into account in these objectives. For the ones depending on a number of jobs — AVEbsld and number of jobs started — all the jobs of each trace are taken into account.

6.5.4 How better is the proposed algorithm compared to a standard power capping mechanism?

Figure 6.2 depicts the normalized mean utilization for each experimental condition depending on the energy budget. The black line is explained in Section 6.5.6. We observe that $energyBud$ outperforms the other algorithms. $reducePC$ performs better than $energyBud$ when opportunistic shutdown is activated for the former and deactivated for the latter. As expected, when the energy budget is of 49 % or lower, all experiments without opportunistic shutdown have the same results —

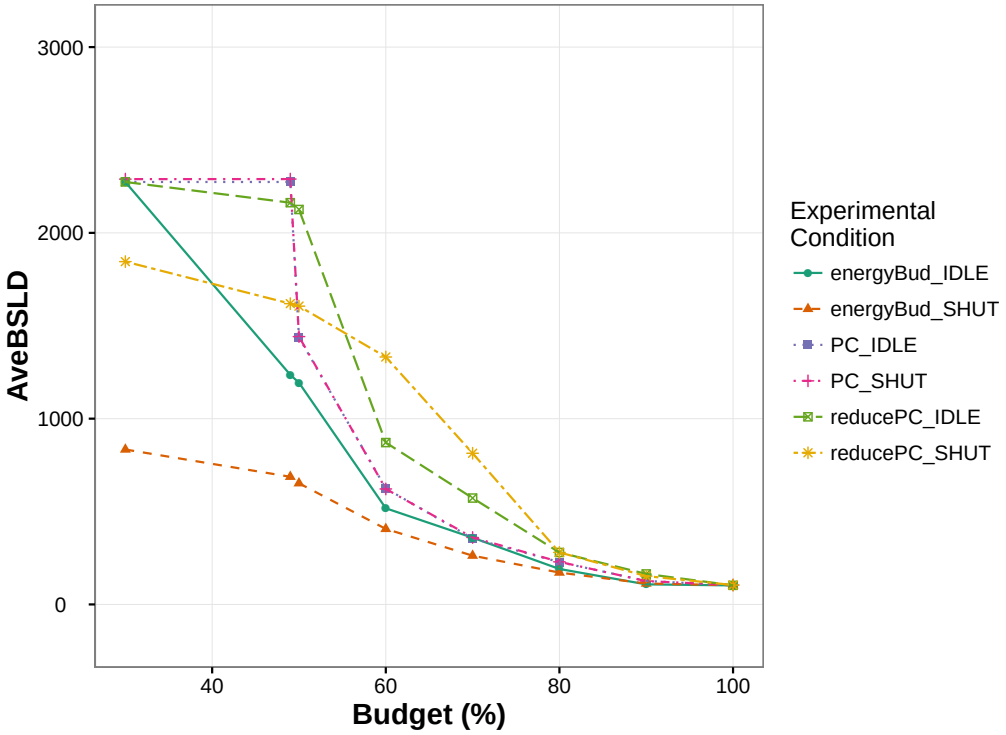


Figure 6.3: Normalized mean AVEbsld for all jobs.

PC_SHUT also has the same performance as the PC mechanism cannot benefit from the energy saved thanks to the opportunistic shutdown.

Figure 6.3 shows the normalized mean AVEbsld for each experimental condition depending on the energy budget. The AVEbsld are very high because we chose traces with a high utilization. In production systems, a high utilization means that a lot of jobs are waiting in the queue. AVEbslds increase even more when the energy budget is low, as resources are limited during a considerable part of the week. Once again, $energyBud$ outperforms all other algorithms — with or without opportunistic shutdown. Surprisingly, the $powercap$ mechanism is not the worst. As expected, the two $powercap$ variants have close results since the algorithm does not benefit from a reduced energy consumption.

Figure 6.4 presents the normalized mean energy consumed during the week relative to the total energy consumable during the same period. When the energy budget is set to 30 %, the algorithms without opportunistic shutdown consume more energy than the total energy consumable — as a 30 % budget is below the energy consumption of a fully idle cluster. The cluster clearly consumes less energy when opportunistic shutdown is on. $energyBud$ consumes more energy than $reducePC$, which consumes more energy than $powercap$. Our algorithms do not try to minimize

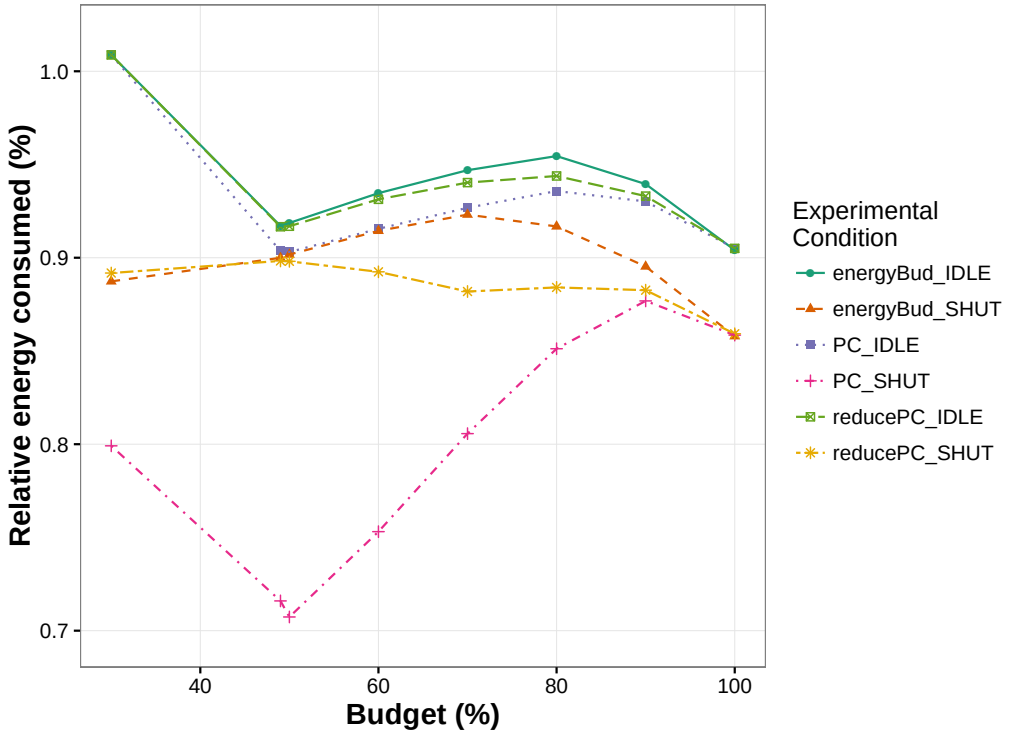


Figure 6.4: Normalized mean energy consumed during the week relative to the maximum energy consumable during the same period.

the energy consumption but try to keep it under a certain value. This behavior can be observed in figure 6.4: $powercap$ has a very low energy consumption that correspond to a non-utilization of the energy saved. At the opposite, $energyBud$ is quite successful at using saved energy as it has a high utilization while having a high energy consumption.

6.5.5 What do we gain by employing opportunistic shutdown?

Table 6.2 shows the average performance difference of different objectives when the opportunistic shutdown mechanism is employed. This table has been computed by taking every experimental condition *with* opportunistic shutdown and by comparing it against its idle counterpart. The comparison is done by computing the percent change of opportunistic shutdown over idle: $(y_{SHUT} - y_{IDLE})/y_{IDLE}$, where y_{SHUT} is a normalized measure with opportunistic shutdown activated and y_{IDLE} the normalized measure in the exact same experimental setting — same algorithm and same budget — as y_{SHUT} but without opportunistic shutdown activated. This table presents the average of these computations.

	Percentage change		
Measure	$powercap$	$reducePC$	$energyBud$
AVEbsld	0.16 %	0.88 %	-8.61 %
Utilization	-0.05 %	4.95 %	5.74 %
Number of job started	-0.05 %	1.4 %	1.47 %
Energy consumed	-4.74 %	-1.78 %	-1.42 %

Table 6.2: Average improvements on different objectives when opportunistic shutdown is enabled. For AVEbsld and Energy, negative values are better.

As expected, activating the opportunistic shutdown mechanism decreases the energy consumption of the $powercap$ algorithm. Activating opportunistic shutdown also decreases the energy consumption of the two other algorithms but less because the saved energy is used to launch more jobs. $energyBud$ takes the most of the opportunistic shutdown. While $reducePC$ and $energyBud$ increase the number of jobs started by the same amount, $energyBud$ improves far more the utilization and AVEbsld. Even more, the activation of opportunistic shutdown dramatically reduces the AVEbsld of $energyBud$.

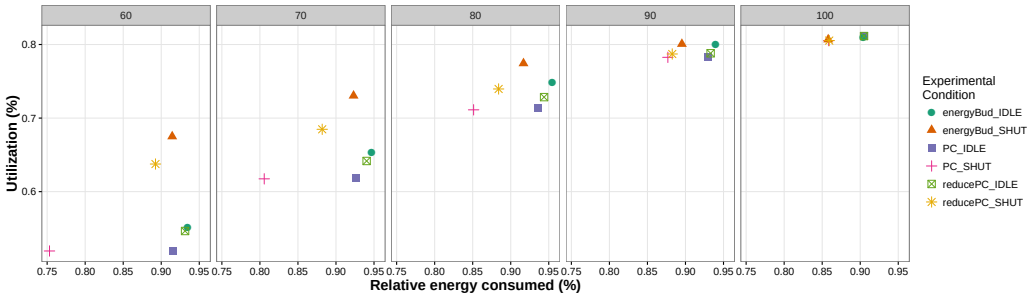
6.5.6 Does reducing the budget to 80% lowers performances to 80%?

In figure 6.2 the black line represents a theoretical performance baseline. If the energy budget is reduced by a certain amount, one can expect the performance to decrease by the same amount. This is what this line represents. The line is not the identity because the energy budget only lasts 3 days during the 7 days considered. Thus, this theoretical performance baseline is formulated as:

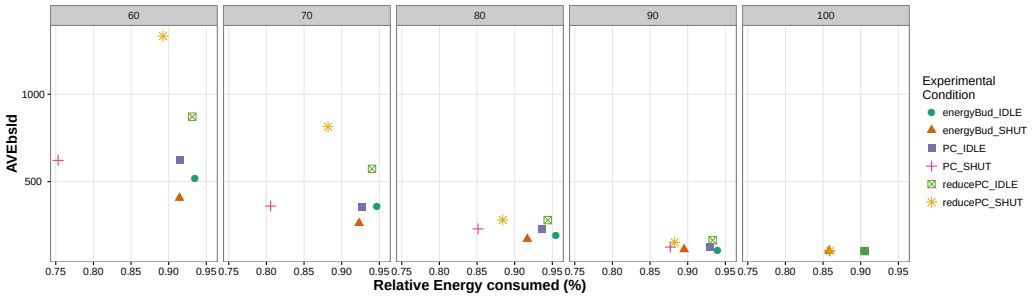
$$f(budget) = \bar{u}_{EASYbf}.(\frac{3}{7} \times budget + \frac{4}{7})$$

where $\bar{u}_{EASYbf}$ is the mean normalized utilization when running the standard EASY backfilling algorithm. If a point is below this line, it means that the performance has decreased more than the energy budget have been decreased.

All the points are surprisingly above the line when the energy budget is set to 90 %. It means that a better energy efficiency is achieved than EASY backfilling — even with a simple $powercap$ mechanism. Presumably, the small limitation in energy reduces the fragmentation and thus improves the utilization. For $energyBud$ with



(a) Normalized mean utilization against normalized mean relative energy consumed for 5 energy budgets.



(b) Normalized mean AVEbsld versus normalized mean relative energy consumed for 5 energy budgets.

Figure 6.5: Performance/energy trade-offs against energy budget.

opportunistic shutdown, this is also true for energy budgets of 60% and above. Without opportunistic shutdown, this is only true for 80% and above.

We can take a look at the trade-offs shaped by the different algorithms to go further in this analysis. Figure 6.5a shows the normalized mean utilization versus the normalized relative mean energy consumption for each experimental condition for different energy budgets. The best points in term of energy-performance trade-off are the most upper left ones. A clear difference appears when opportunistic shutdown is activated: The points are in the upper left part of the graphs. $energyBud$ with opportunistic shutdown has the best trade-off. $powercap$ is on the Pareto curve but with a very low utilization.

Trade-offs between the normalized mean AVEbsld and the normalized relative energy consumed are shown on figure 6.5b. Here the best points are the lower left ones. Once again $powercap$ with opportunistic shutdown has a good trade-off because of its low energy consumption. However, $energyBud$ with opportunistic shutdown has the best AVEbsld trade-off. If we only look at the points without opportunistic shutdown, $powercap$ and $energyBud$ have the best energy/AVEbsld trade-offs.

6.5.7 Which one is the best between reducePC and energyBud?

We consider $energyBud$ as the best of the two proposed algorithms. This algorithm provides the best utilization and AVEbsld results. Even more, we have seen that using this algorithm for not so low energy budget increases the energy efficiency of the cluster compared to the standard EASY backfilling.

6.6 Conclusion

The purpose of this work was to extend the widely-used EASY backfilling algorithm to comply with periods during which energy availability is limited.

We proposed two new alternatives and showed their effectiveness on a wide range of scenarios that have been assessed through simulation. These two new algorithms not only provide a way to control the energy consumption of computing platforms but also optimize metrics such as system utilization and bounded slowdown. Moreover, the proposed algorithms improve the energy efficiency of the cluster when the amount of available energy is large.

As this work is an improvement of EASY backfilling, our algorithms still support most existing extensions of this algorithm, such as advanced reservations, preemption mechanisms or the establishment of a maximum power limit.

As future work we would like to implement our algorithm upon a real open-source resource and job management system such as Slurm or OAR and study its effects in a supercomputer in production. One limitation of our approach is that we only considered periods with a fixed energy budget. Hence, this work could be extended to become more dynamic: If the energy budget followed electricity price, we could control and thus reduce a significant part of the cluster costs. This would provide an improved solution to the remaining use cases described in [Pat+16a] where the electricity cost varies as it partially depends on renewable sources.

7 Energy vs Responsiveness Trade-off in EASY Backfilling

7.1 Introduction

While the fastest High Performance Computing (HPC) systems are evolving to extreme-scale platforms, there is a large adoption of cluster and cloud computing by all sorts of industries for their production needs. In contrast to the research centers at the forefront of innovation with an almost unlimited budget, these companies do not attempt to leverage the most floating point operations per second out of their clusters to be ranked first in a global race but to efficiently spend their computing budget to get the most out of their necessarily limited needs.

One of the most promising way of reducing the expenses tied to a cluster is to reduce its power consumption. To begin with, the power consumption becomes a constraint since electricity companies set upper bounds on the power that they can provide for clusters at different periods of time. Actually the problem not only lies in the available power but also on the huge energy cost — several years of intensive use may be greater than the price of the platform itself. It is therefore mandatory to develop efficient tools to utilize new HPC clusters at a sustained performance rate with a lower energy consumption.

While speed-scaling is the natural way to save energy while running a job, switching down nodes is the natural way to save energy between jobs. However, most of the existing resource management algorithms focus only on performance and energy savings are only an afterthought. Taking into account node shutdowns at the resource manager level while scheduling the jobs can potentially yield significant improvements, as idle periods can be lined up to turn off fewer nodes for longer periods of time.

Thus, we propose in this work to develop new methods for enhanced job allocations, unlocking a potential source of energy savings — arguably the most promising one — by creating better shutdown opportunities at the job scheduling level.

We consider this problem as a bi-objective optimization whose purpose is to determine a good trade-off between a maximum number of switched-off processors while keeping a good performance. Performance will be measured as it is common by minimizing the average waiting times — this objective is defined in section 7.2.3 for the sake of convenience and also in the main notations chapter in section 2.4.

Given this setting, we designed and compared algorithms based on two energy saving methods. The first is an extension of the regular EASY backfilling scheduling algorithm, where idle nodes are switched-off whenever they have been idle for a while, and started again if they can be used for new jobs. The second method is more actively saving energy by planning to turn off some nodes depending on the current state of the schedule and of the queues, and keeping those nodes off as long as it does not deteriorates our performance measure too much. As can be seen from this first description, many parameters can be adjusted. To keep this chapter as clear as possible, only the most meaningful results are presented.

The assessment of the proposed algorithms on real execution traces in simulation shows that significant energy savings (up to 25%) can be achieved. The most interesting result is that using the second method described above these savings were possible with a relatively limited number of switches.

Chapter content. The chapter is organized as follows. A precise description of the problem is given in section 7.2. Then section 7.3 describes the energy saving techniques employed by our algorithms. Section 7.4 proposes and presents several algorithms. The simulation campaign is reported in section 7.5 and its results are analyzed in section 7.6. Related works are given in section 7.7. Finally, we conclude and give future works in section 7.8.

7.2 Problem Description

Distributed platforms generally follow the same logic where computationally intensive jobs are submitted in waiting queues. Then, the *resource and job management system* — RJMS in short — collects data on these submitted jobs, analyzes them and finally determines an allocation according to the available resources.

7.2.1 Job characteristics

We consider parallel and rigid [Fei15] jobs that are submitted on-line. It is common for the queue to be sorted according to the time when the jobs are submitted (First-Come-First-Served, denoted shortly by FCFS).

Jobs are indexed by j and are characterized by their release time (denoted by r_j), their number of requested computing resources (q_j), and their wall-time ($wall_j$). Both q_j and $wall_j$ are specified by the users and are not known to the scheduler before r_j. The processing time (denoted by p_j) remains unknown until the job is completed. Job j is killed if it reaches its wall-time, without any penalty on the scheduler (in this case we simply define p_j as equal to $wall_j$). The jobs are not preemptive, which means that once started a job cannot be interrupted until its completion.

Once the jobs have been executed more information about them can be defined. We denote by $start_j$ the time at which j starts being executed, and by C_j the time at which it completes ($C_j = start_j + p_j$). $wait_j = start_j - r_j$ denotes the amount of time j stayed in the system before being executed.

7.2.2 About the platform

We focus on homogeneous computing platforms, whose computing resources will be simply referred to as *machines*. A platform is defined by a set of m machines. The interconnection network is not explicitly taken into account in this chapter.

The machines have various states $S = \{\ computing, idle, off, on \rightarrow off, off \rightarrow on\ \}$. Idle machines can be switched-off, which takes a time $t_{on \rightarrow off}$. Off machines can be switched-on, which takes a time $t_{off \rightarrow on}$. At a given time t, machine i is in one and only one of the previous states.

Since in the HPC context the machines are affected to unique jobs, only $idle$ machines can be selected to compute new jobs. An instantaneous power consumption is associated to each state, denoted by $P_s\ \forall s \in S$, and it is expressed in watts.

The electrical power consumption of a machine is entirely defined by its state. With our simulation tool, it is pretty straightforward to include many different computing states to reflect the dynamic voltage and frequency scaling mechanism (DVFS). However, in order to have a fair assessment of the improvements brought by our policies, we considered that for every simulation the jobs had the exact same power profile — i.e., every job has the same duration and power consumption regardless

of when and where it is scheduled. Therefore, precisely modeling DVFS would only change a constant fraction of the power consumption for every simulation.

7.2.3 Objectives

We are interested in the following bi-objective optimization problem: minimize both the system unresponsiveness and the consumed energy of the platform. As this is a bi-objective problem, we will provide a set of solutions that are not completely comparable (Pareto set) [Dut+09].

What we call the *responsiveness* of a system represents its ability — in a given state — to start jobs rapidly. We selected two metrics that represent the opposite of this notion, that is the *unresponsiveness*. First, the jobs mean waiting time is defined by $\frac{1}{n}\sum_j wait_j$, and is expressed in seconds. We are also interested in the maximum waiting time, defined by $\max_j wait_j$ and also expressed in seconds.

The total consumed energy of the platform is defined as the sum of the consumed energy of its machines, and is expressed in joules. The energy consumed by a machine is defined as usual as the integral of its power over time. The time boundaries which interest us here are between the submission time of the first job and the completion time of the last one.

This bi-objective problem is to determine a trade-off — as both objectives are conflicting. More precisely, the solutions that do not worsen too much the system unresponsiveness are those which are the most relevant. Since the problem is most likely NP-hard and that the targeted sets of jobs and platforms may be huge, we are not looking for Pareto optimal solutions but for reasonable and useful alternatives.

7.3 Energy Minimization Techniques

Various techniques are used in the literature to save energy. These techniques can be split in two parts. First there are techniques that aim at reducing the energy consumption of the jobs themselves. Second, they focus on a better resource exploitation to avoid energy wastes. In this chapter, we consider that the first kind of techniques are fully used at the job level and we choose to study only the second kind of techniques to improve their understanding.

7.3.1 DVFS

The dynamic voltage and frequency scaling (DVFS) mechanism — also known as speed-scaling — is probably the most studied technique to reduce the jobs energy consumption, especially in theory [Alb10]. It can be used to ensure that power-limited systems do not exceed a power threshold [GGT15]. Using DVFS at the platform management level is risky as it can increase the overall energy consumption of some applications [CM10].

In this chapter, we suppose that jobs are energy efficient. It means that we consider that DVFS decisions are done within the jobs and not at the resource management level. This approach may reduce the energy optimization space, but it also prevents bad DVFS decisions that might result in increased jobs energy consumption [SRH05].

7.3.2 Exploiting Idle Time

Idle machines consume a lot of energy on current hardware. As this part of the energy is not used for computing jobs, we consider it as lost. Hence, reducing this part of the energy consumption is crucial and a big source of gain.

One way to save energy is to switch idle machines into lower consumption states. This mechanism greatly reduces the machines idle power consumption but it should be used with caution as switching to and from lower consumption states has both time and energy costs, and potentially ages the hardware.

Lower consumption states can be different types of sleep states or complete shutdowns. For the sake of readability, we will simply refer to machines in lower consumption states as off, but please notice that using other lower consumption states does not change the approach at all.

Opportunistic Shutdown

What we call opportunistic shutdown is a type of machine shutdown technique. It keeps track of the machines states over time. Whenever a machine remains idle for more than t_{idle} seconds, it is switched-off. Some constraints can be added into this technique to prevent unintended behaviors — e.g., preventing it to hinder priority jobs.

We propose in this chapter to use quite directly the machines that have been switched-off *via* this technique. The resource management algorithm can use these resources to compute jobs freely, the machines just need to be switched back to a computing state before actually executing the jobs. To limit the number of state switches and to minimize the system unresponsiveness, idle machines should be selected with a higher priority than off ones.

Off Reservations

Another approach is to deliberately reserve some machines as off. Contrary to opportunistic shutdown, off reserved machines cannot be used freely to compute jobs, the off reservation must first be cancelled or altered. Even if this approach may conflict with the performance objective, we advocate that it allows worthwhile energy and performance trade-offs if used wisely.

In this chapter, we will test two ways to use off reservations. First, we will consider a static view where a fixed proportion of machines is to be used as little as possible. In this naive approach, off reserved machines can only be switched-on if a job do requires them, that is to say the job cannot be executed at all without these machines. Second, we will consider to adapt the number of off reserved machines to the system responsiveness. For this purpose, we will propose an online estimator of the system unresponsiveness.

7.4 Algorithms

This section describes the proposed algorithms to address the problem described in section 7.2. These algorithms make use of the techniques introduced in section 7.3. Preliminary notions are given in section 7.4.1, the algorithms themselves are detailed respectively in sections 7.4.2 and 7.4.3, implementation details are mentioned in section 7.4.4 and the following paragraph defines the decision space of the algorithms.

The algorithms studied in this chapter are event-based. The events may concern job submissions, job completions, or state modifications of machines — e.g., some machines just finished to be switched-off. Whenever they are called, the algorithms take decisions, which are performed immediately after the decision making. Decisions include starting the execution of a job on some machines, or switching the state of some machines — e.g., switch-on some machines.

7.4.1 Preliminaries

Easy Backfilling

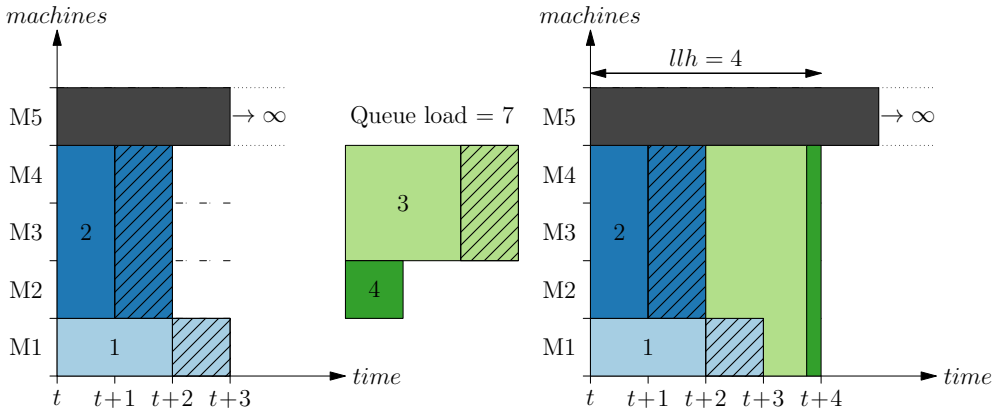
Easy backfilling [MF01], that will simply be referred to as EASY in the remainder of this chapter, is one of the mostly used resource management algorithm. This popularity can be explained by the ease of implementation and the high utilization its aggressive backfilling mechanism induces, since most HPC center administrators often want to maximize the resource utilization.

EASY stores the pending jobs in a queue, which is ordered by job arrival time. First, EASY scans the job queue in order as long as the jobs can be executed right away. This part of the algorithm is the same as the First Come First Serve policy (FCFS). If a scanned job cannot be executed right away, a reservation is done for it and the queue scanning is stopped. Such a job is called a *priority* job. Second, EASY scans the remaining jobs in the queue (still in order) and attempts to *backfill* them — i.e., execute them forthwith if and only if they do not seem to delay the starting of the priority job.

Unresponsiveness Estimator

As stated before, what we call the responsiveness of a system represents its ability — in a given state — to start jobs rapidly. Instead of working directly on the responsiveness, we propose an estimator of the opposite of this notion, that is the *unresponsiveness* estimator of the system at time t, and is denoted by $\upsilon(t)$.

More concretely, we propose a system unresponsiveness estimator called the liquid load horizon (LLH). It estimates the needed amount of time to compute the pending load in the queue. For this purpose, it first computes the total pending load via $l = \sum_j q_j \cdot wall_j$. This estimator then assumes that this load l can be distributed freely among the computing machines. The value computed by this estimator is the needed amount of time to consume the load l in the provisional schedule — which depends on the running jobs, on the machine switches currently ongoing, and on the current state of the machines. This calculation is represented on figure 7.1. Computed values are in range $[0, +\infty[$. The degenerated case $+\infty$ can be obtained when all machines are switched — or are switching — off. The variation of the LLH over time is a piece-wise linear function with jump discontinuities on events, and is decreasing between events.



(a) Provisional schedule at time t: jobs 1 and 2 are being computed, machine 5 is switched-off. Jobs 3 and 4 are in queue.

(b) The liquid load horizon of this system is 4, as it takes 4 seconds to spread the total load (in blue and pink) among the different machines.

Figure 7.1: The liquid load horizon, a system unresponsiveness estimator. Black: switched-off. Dashed areas is the difference between predicted and actual job processing times.

The LLH is subject to uncertainties about times, as jobs execution times and machines switching times are unknown in advance. It also considers the ideal situation in which the load can be divided freely among the machines.

7.4.2 Algorithm 1: Proportional Shutdown

The *Proportional Shutdown* algorithm is an extension of EASY — cf. section 7.4.1. The main idea of this algorithm is to keep a proportion of the machines in an off state most of the time. In other words, this algorithm makes an initial off reservation and only reduces its size when big jobs (those requesting too many machines) must be executed.

The only parameter specific to this algorithm is $\rho \in]0, 1]$, which defines the proportion of machines that can be used most of the time. The number of machines used most of the time is $\lfloor \rho \cdot m \rfloor$. This constraint is only violated when the priority job requests more machines than $\lfloor \rho \cdot m \rfloor$. This algorithm also supports the opportunistic shutdown technique, which switches-off machines that remained idle longer than t_{idle} seconds.

It may be noticed that EASY is a special case of this algorithm when $\rho = 1$ and when the opportunistic shutdown is disabled. Furthermore, when $\rho = 1$ and that the opportunistic shutdown is enabled, the resulting instances are equivalent to EASY with an opportunistic shutdown technique.

Algorithm 3: Inertial Shutdown periodic decision-making procedure

Input: dt_{t-T}, the type of the previous energy decision
$\#switch_{t-T}^{real}$, the actual number of switched machines since $t-T$
$\bar{v}_t$ and $\bar{v}_{t-T}$, the mean of the LLH over $]t-T,\ t]$ and $]t-2\,T,\ t-T]$
$\bar{v}_{ub}$, the threshold to control the unresponsiveness increase
Output: dt_t, the type of the energy decision to take
$\#switch_t^{due}$, the number of machines to switch

```
if v̄_t ≥ v̄_ub then
    if dt_{t-T} = SWITCH OFF then
        dt_{t-T} ← SWITCH ON
        #switch^{real}_{t-T} ← 0
    end
    v̄_{t-T} ← 0
end
if ((dt_{t-T} = SWITCH ON) and (v̄_t > v̄_{t-T})) or
  ((dt_{t-T} = SWITCH OFF) and (v̄_t ≤ v̄_{t-T})) then
    dt_t ← dt_{t-T}
    #switch^{due}_t ← min( max(f(#switch^{real}_{t-T}), 1), μ )
else
    dt_t ← ¬dt_{t-T}
    #switch^{due}_t ← 0
end
```

7.4.3 Algorithm 2: Inertial Shutdown

The *Inertial Shutdown* algorithm is another EASY extension — cf. section 7.4.1. Its main idea is to adjust the number of usable machines to the unresponsiveness variation of the system. In other words, this algorithm maintains an off reservation whose size is completely dynamic.

To do so, this algorithm uses the LLH system unresponsiveness estimator introduced in section 7.4.1. It is also compatible with the opportunistic shutdown technique.

This algorithm makes energy decisions periodically, which limits the computations to do at each event and allows to react when no event occur — e.g. when long jobs are being computed. Algorithm 3 explains this decision-making procedure, which is called at time t and decides the type of decision to make dt_t — either to SWITCH ON or to SWITCH OFF — and the number of machines to switch $\#switch_t^{due}$. The switches are then initiated now if possible or as soon as possible otherwise. The following paragraphs of this section detail the different aspects of algorithm 3.

To find out whether the system unresponsiveness is increasing or decreasing, this algorithm compares representative values of LLH over the current period $]t-T,\ t]$ and the previous one $]t-2\,T,\ t-T]$. We used the mean of LLH over the period as its representative value, denoted by $\bar{v}_t$ and detailed in equation (7.1), where t is the current time and T is the period duration.

$$\bar{v}_t = \frac{1}{T}\int_{t-T}^{t} v(x)\ dx \tag{7.1}$$

This algorithm keeps track of the previous energy decision it took (at time $t-T$) and whether the related switches have been initiated. dt_{t-T} is the previous decision type and $\#switch_{t-T}^{real}$ is the number of machines whose switching has been at least initiated since $t-T$. Please do note that $\#switch_{t-T}^{real}$ does not include the switches resulting from the opportunistic shutdown technique — i.e., $\forall t, \#switch_t^{real} \leq \#switch_t^{due}$.

We introduced a threshold denoted by $\bar{v}_{ub}$ to control how high the system unresponsiveness is allowed to grow. If, at time t, $\bar{v}_t \geq \bar{v}_{ub}$, some machines are switched on right away regardless of the previous decision type.

This algorithm continues to take the same type of decision if the LLH variation remains of the same sign. Explicitly, if the algorithm decided to switch some machines on (off) previously and that the LLH has increased (decreased), it decides to switch machines on (off). Otherwise, the algorithm toggles its decision type but does not take any decision instantly to avoid overreacting. The algorithm computes $\#switch_t^{due} = \min\Big(\max\left(f(\#switch_{t-T}^{real}), 1\right), \mu\Big)$ where μ is the number of switchable machines (now or in expected future) and f is a (not strictly) increasing $\mathbb{N} \to \mathbb{N}$ function. Function f has been introduced to allow variations of the switches aggression and should allow to control the algorithm inertia.

7.4.4 Implementation Issues

This section gives details about how we implemented the proposed algorithms. As we promote research reproducibility, feel free to read our implementation [@batsched] if some parts remain unclear despite our best efforts.

In our implementation, the two algorithms share the same code base to handle *usual* events — i.e., job submissions, job completions and machine switches. The algorithms only differ by their decision-making procedure when periodic calls occur. This code base implements a modified version of EASY that allows to handle machine

switches, the marking of machines as waiting to be switched, and the opportunistic shutdown technique.

This code base also stores how many machines have been switched for two types of reasons, either for being idle or because the algorithm decided not to use these machines. When the algorithms decide to increase the off reservation size, if machines have already been switched-off because of idleness, those machines are *stolen*. This avoids to switch-off new machines for the purpose of minimizing the number of switches. Whenever an algorithm marks some machines as waiting to be switched it selects the soonest switchable machines according to the provisional schedule.

The code base uses a machine priority in its backfilling procedure. Idle machines are used with a higher priority than the machines that were switched-off by the opportunistic shutdown technique. Off reserved machines are only switched-on if the priority job cannot be executed: When it requires more machines than those currently on + those being switched-on + those currently off because of the opportunistic shutdown technique.

Finally, please remark that the parameters of the two algorithms are dynamic and can be adjusted at any time by a system administrator — even if it is not done in the experiments. Furthermore, the two algorithms do not strongly depend on the periodic call mechanism detailed in this section, as any callback or timeout mechanism may be used instead.

7.5 Evaluation Process and Reproducibility

We established a benchmark in order to assess the behavior of the algorithms presented in section 7.4. The aim of this benchmark is to characterize the algorithms performance — on both objectives — according to their parameters, and to know whether one technique or combination of techniques is clearly better than the others. Since research reproducibility is a major concern for us, our evaluation process is detailed in the present section. Most of the required resources to replicate this experiment can be found on repository [@reprETO17]. This includes the controlled software environment which has been set up for this experiment thanks to Kameleon [Rui+15], the various scripts used to produce the data and to analyze it, an aggregated copy of our experimental data, and finally some figures that have not been selected for this paper.

The algorithms are assessed in simulation thanks to Batsim. All the experiments have been executed with Batsim commit 709c160 of repository [@batgit2] and commit 3aaa179 of our algorithms repository [@batsched].

The EASY, proportional shutdown and inertial shutdown algorithms have been executed in various situations. These situations, that will be denoted as *instances*, are defined by the used algorithm, a combination of the algorithm parameters, the executed workload and the simulated platform. The parameter space of our experiment is summarized in table 7.1. EASY is not shown is this table as it does not have any parameter. The different combinations leads to 1282 instances, which totals to more than 1800 years of simulated time and nearly 50 million jobs.

Table 7.1: Parameter space of our experimental process. All combinations are explored.

Shared by all algorithms	
Workloads	KTH_SP2, SDSC_SP2
Platform	homogeneous240
Shared by Proportional and Inertial	
T (s)	60, 120, 300, 600
t_{idle} (s)	0, 300, 600, 6000, $+\infty$
Make run decisions on period	true, false
Proportional-specific	
ρ	1.00, 0.95, 0.90, 0.85
Inertial-specific	
$f(n)$	$n+1$, $n \times 2$
$\bar{v}_{ub}$ (s)	$1 \cdot 10^4$, $1 \cdot 10^5$, $2 \cdot 10^5$
Allow future switches	true, false

Since we are interested in fully homogeneous platforms in this chapter, we have generated a simple SimGrid platform — a cluster of 240 computing machines. This platform is used to compute all the instances, but only a subset of the computing machines can be used — depending on the workload that is being executed. Energy data of the computing machines comes from a series of measures conducted on the Taurus Grid'5000 test-bed as detailed in section 6.5.1. The machines energy parameters are shown in table 6.1 (page 68).

As we are interested in the utilization variation over time of real platforms, and since we would like to evaluate how our algorithms behave in realistic scenarios, we chose to evaluate them on workloads coming from real traces. We chose to execute them entirely in a single block — rather than cutting them is different slices — which allows to preserve the various utilization cycles — e.g., daily and weekly — and to preserve some characteristics of these workloads. We chose to use the KTH_SP2 and SDSC_SP2 workloads from the Parallel Workload Archive [FTK14;

@pwa]. These two workloads are long (11 and 24 months), exhibit cyclic utilization patterns and use a relatively small number of machines (100 and 128), which allows to keep relatively short simulation times (in the order of the dozen of minutes for each instance). We used the cleaned versions of these workloads and added another cleaning step that discards jobs with invalid or unspecified execution time or wall-time. Since the network is ignored in this study, the Batsim jobs generated from these workloads only consist in an amount of computation, distributed homogeneously between the computing machines allocated to each job.

In order to assess the behavior of the opportunistic shutdown technique with our two algorithms, various values of the T and t_{idle} parameters have been considered. We also tested whether calling the *usual* decision making procedure after each periodic decision making procedure call altered the algorithm behavior.

The proportional shutdown algorithm has only one specific parameter: ρ, the proportion of machines that must remain on most of the time. Various values between 100 % and 85 % have been tested.

More parameters have been tested for the inertial shutdown algorithm. First, three values of $\bar{v}_{ub}$ have been tested. These values have been chosen by looking at the evolution of the LLH on schedules resulting from EASY for the two workloads. Preliminary results have shown us that the f function impact seemed negligible on platforms of this size, we then only evaluated one linear and one geometric function in this experiment. Finally, we tested whether forcing the inertial switching decisions to be made at periodic calls or to allow them to be conducted later on changed the algorithms results.

7.6 Results

This section shows and analyzes the results of the experiment described in section 7.5. Section 7.6.1 gives the instance nomenclature used in the remainder of this chapter. A result overview of the most interesting trade-offs is given in section 7.6.2. Section 7.6.3 analyzes how much energy can be saved if great responsiveness losses are allowed. Section 7.6.4 proposes a finer grain analysis of our results. Finally, the impact of the most important parameters of each technique and their combination is briefly explained in sections 7.6.5, 7.6.6 and 7.6.7.

7.6.1 Instance Naming Convention

This section makes explicit the instance naming convention we chose to outline our results. The **EASY** instances correspond to the EASY algorithm alone (without any parameter). The **OS** instances correspond to the EASY algorithm used with the opportunistic shutdown technique enabled (formally proportional shutdown with $\rho = 1$ and $t_{idle} \leq 600$). The **weakOS** instances correspond to the EASY algorithm used with a very weak aggressiveness (formally proportional shutdown with $\rho = 1$ and $t_{idle} = 6000$).

The **inertial** instances correspond to the EASY algorithm used in conjunction with the inertial shutdown technique, and with opportunistic shutdown disabled or used with a very weak aggressiveness (formally inertial shutdown with $t_{idle} \in \{6000, +\infty\}$). The **inertial+OS** instances correspond to the EASY algorithm with both opportunistic shutdown and inertial shutdown techniques enabled (formally inertial shutdown with $t_{idle} \leq 600$).

The **prop** instances correspond to the EASY algorithm used in conjunction with the proportional shutdown technique, with opportunistic shutdown disabled or used with a very weak aggressiveness (formally proportional shutdown with $\rho < 1$ and $t_{idle} \in \{6000, +\infty\}$). The **prop+OS** instances correspond to the EASY algorithm used with both proportional shutdown and opportunistic shutdown enabled (formally proportional shutdown with $\rho < 1$ and $t_{idle} \leq 600$).

For the sake of readability, consumed energy is normalized by EASY results for each workload ($2.73 \cdot 10^{11}$ J on KTH_SP2 and $7.66 \cdot 10^{11}$ J on SDSC_SP2).

7.6.2 Most Interesting Trade-offs

As we said before, the solutions that do not worsen much the system responsiveness are the most relevant. This section focuses on these solutions, which are plotted on figures 7.2, 7.3 and 7.4. They have been obtained by setting $\bar{v}_{ub} = 10^4$ for the inertial shutdown algorithm — *inertial* and *inertial+OS* instances — and $\rho = 1$ for the proportional shutdown algorithm — *OS* and *weakOS* instances.

Both inertial and opportunistic shutdown, as well as their combination, allows energy savings of about 20 % for the two workloads — as seen on figure 7.2. The inertial technique allows greater energy savings than the opportunistic one on the KTH SP2 workload, but the opposite phenomenon occurs on the SDSC SP2 workload when the opportunistic shutdown is used aggressively — *OS* instances. Using the

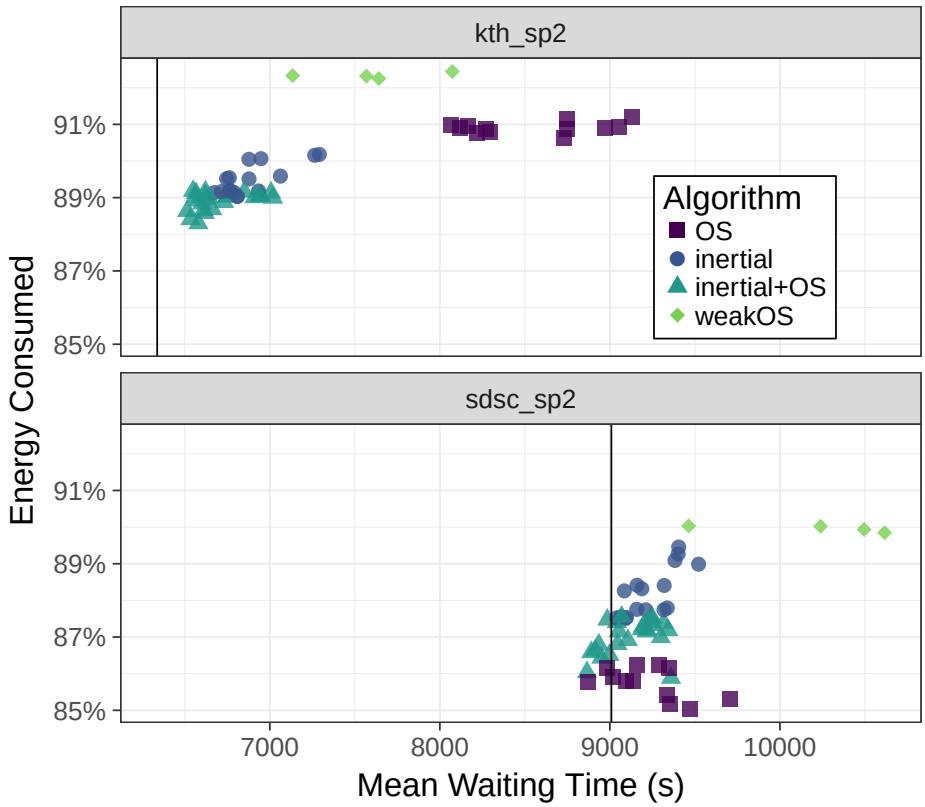


Figure 7.2: Energy vs Mean Waiting Time of the best trade-off solutions (reasonably low waiting time) for the KTH SP2 and SDSC SP2 workloads.

opportunistic shutdown with a weak aggressiveness does not seem very interesting as these instances do not contain any (energy, mean waiting time) Pareto-optimal solution in the explored parameter space. Enabling the opportunistic shutdown technique slightly improves inertial (energy, mean waiting time) trade-off results on both workloads.

The inertial technique allows lower mean waiting times globally and is more stable — the points are more compact on this metric — than opportunistic shutdown, as seen on figure 7.3. The inertial technique is clearly better on the maximum waiting time metric on both workloads — results are only presented on SDSC SP2 but results are even better on KTH SP2. This leads us to conclude that the inertial technique allows to obtain a higher system responsiveness than the opportunistic technique.

The inertial technique does significantly fewer switches than the opportunistic technique, as figure 7.4 shows. This result is important, as increasing the number of switches is suspected to increase the failure rate and to reduce the machine lifespan.

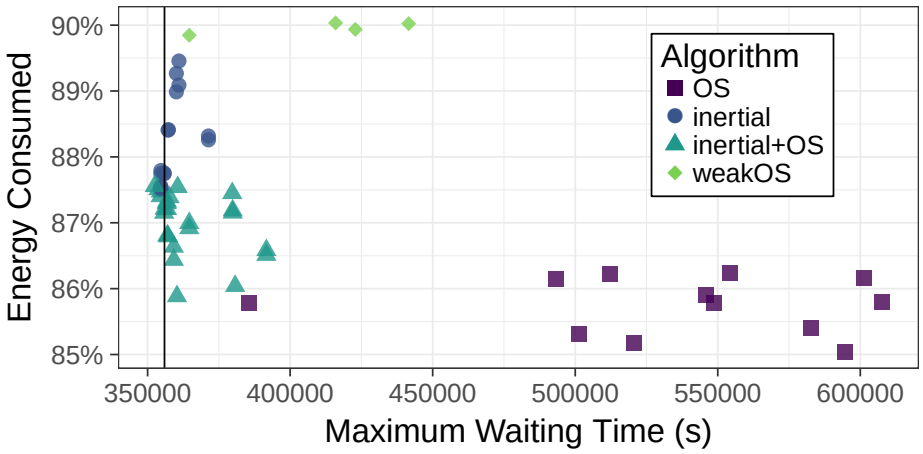


Figure 7.3: Energy vs Max Waiting Time of the best trade-off solutions for the SDSC SP2 workload.

Both inertial and opportunistic shutdown allows to save significant amounts of energy (gains are of the same order for both techniques). However, inertial shutdown avoids high system responsiveness decreases and involves much fewer switches, which leads us to think that the inertial technique is better than the opportunistic one.

7.6.3 All Trade-offs

Another goal of this experiment is to determine whether, thanks to the proposed techniques, considerable energy amounts can be saved if substantial responsiveness losses are allowed. For this purpose, figure 7.5 shows all the trade-off solutions resulting from our experiment. It is a bigger picture of figure 7.2, as it includes *prop* and *prop+OS* instances, all $\bar{v}_{ub}$ values, and multiple combinations of negligible parameters. The negligible parameters are: 1. Whether forcing the inertial switching decisions to be taken at periodic calls or to allow them to be conducted later on 2. Whether to call the *usual* decision making procedure after each periodic decision making procedure call.

The opportunistic technique does not shape clear trade-offs between energy and mean waiting time. On the other hand, both the inertial and the proportional techniques allow to get trade-offs, as increasing the mean waiting time allows more energy savings. However, we can notice that only tiny energy savings can be obtained this way, at the cost of great mean waiting time increases. This led us to

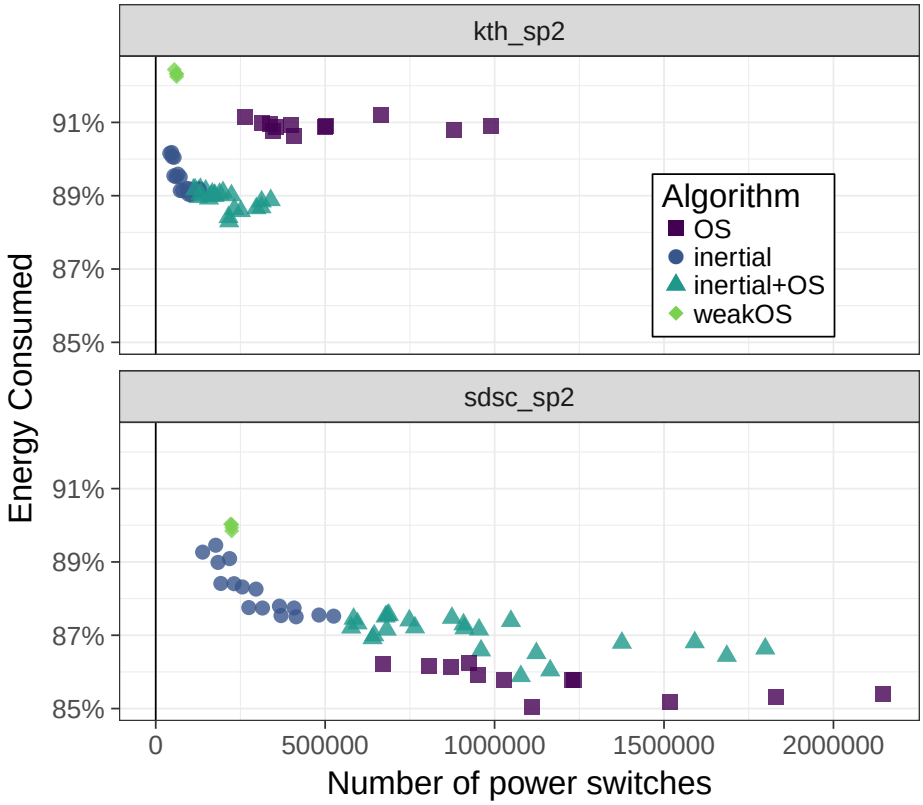


Figure 7.4: Energy vs switch number of the best trade-off solutions (reasonably low waiting time) for the KTH SP2 and SDSC SP2 workloads.

the conclusion that the trade-offs that should be made in practice are those which keep a low mean waiting time, as described in section 7.6.2.

The proportional technique results are plotted on figure 7.5. When used alone (*prop* instances), none of the solutions is Pareto-optimal. When used in conjunction with the opportunistic shutdown technique — *prop+OS* instances — most of the energy savings seems to come from the opportunistic technique. This can be seen on SDSC SP2, where the *prop+OS* instances are split in two *rows*. The upper row (around 80 % energy) comes from *weakOS* instances ($t_{idle} = 6000$), while the lower row (around 75 % energy) comes from the *OS* instances ($t_{idle} \leq 600$). Consequently, this leads us to conclude that the proportional shutdown technique is not very interesting. It is indeed always dominated by the inertial shutdown technique, which allows dynamic variations of the off reservation size.

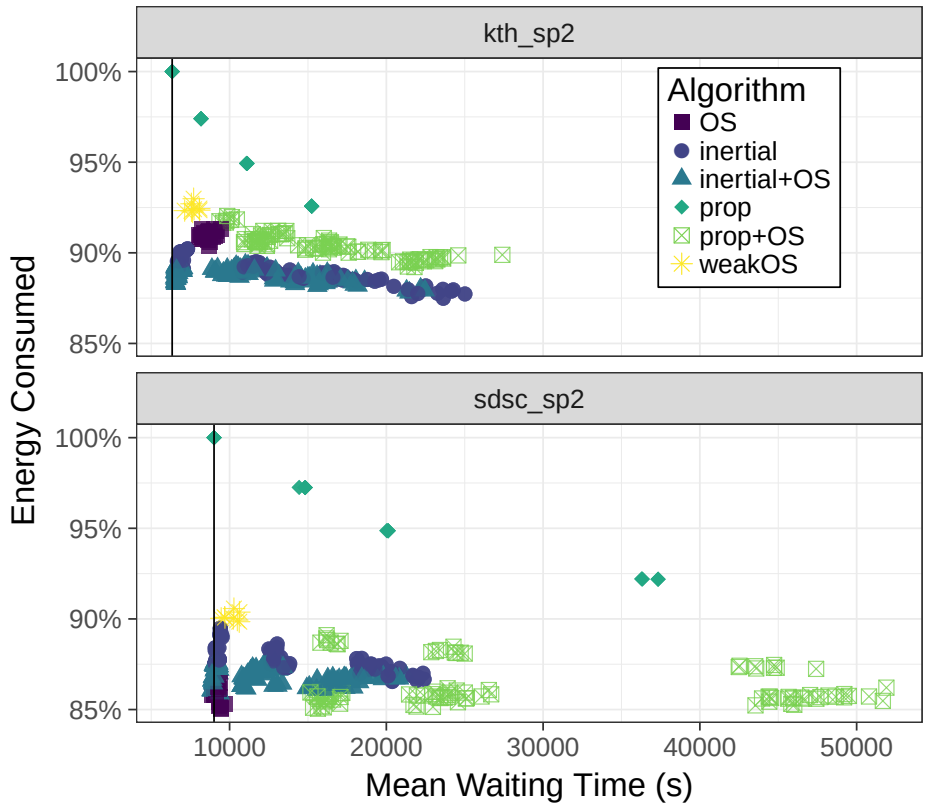


Figure 7.5: Energy vs Mean Waiting Time of all trade-off solutions for the KTH SP2 and SDSC SP2 workloads.

7.6.4 Finer Grain Analysis

As both workloads are long, they exhibit various patterns over time. This section analyzes the results of our experiment by aggregating results by groups of 4 consecutive weeks — instead of aggregating them over the whole workload length. For the sake of conciseness, groups of 4 consecutive weeks will simply be referred to as *months*.

Figure 7.6 shows that the proposed algorithms can in fact achieve greater energy savings than those presented in section 7.6.2, as up to 30% energy savings are shown. These savings mostly depend on the load pressure, as can be seen in the correlation between the mean utilization during a single month and the consumed energy.

The most frequent trade-off patterns that can be observed in the months are explicited in Figure 7.7. Half of KTH months are similar to KTH's month 2, where the algorithms results are ordered (inertial+OS is better than inertial, which is

better than OS, which is better than weakOS). A quarter of KTH months are similar to KTH's month 9, where the algorithms are different trade-offs between energy and performance. 73% of SDSC months are similar to SDSC's month 12, where the algorithms have similar performance but various energy consumption (OS is generally better than inertial+OS, which is better than inertial, which is better than weakOS). 11% of SDSC months are similar to SDSC's month 13, where the results are dense and noisy on both objectives. Finally, SDSC's month 15 is similar to 8% of SDSC and KTH months, where the algorithms show rather close energy savings but potential high performance drops. The latter case exhibits that the opportunistic technique is more likely to take *bad* shutdown decisions in some cases (notably when the system is heavily loaded) than the inertial one.

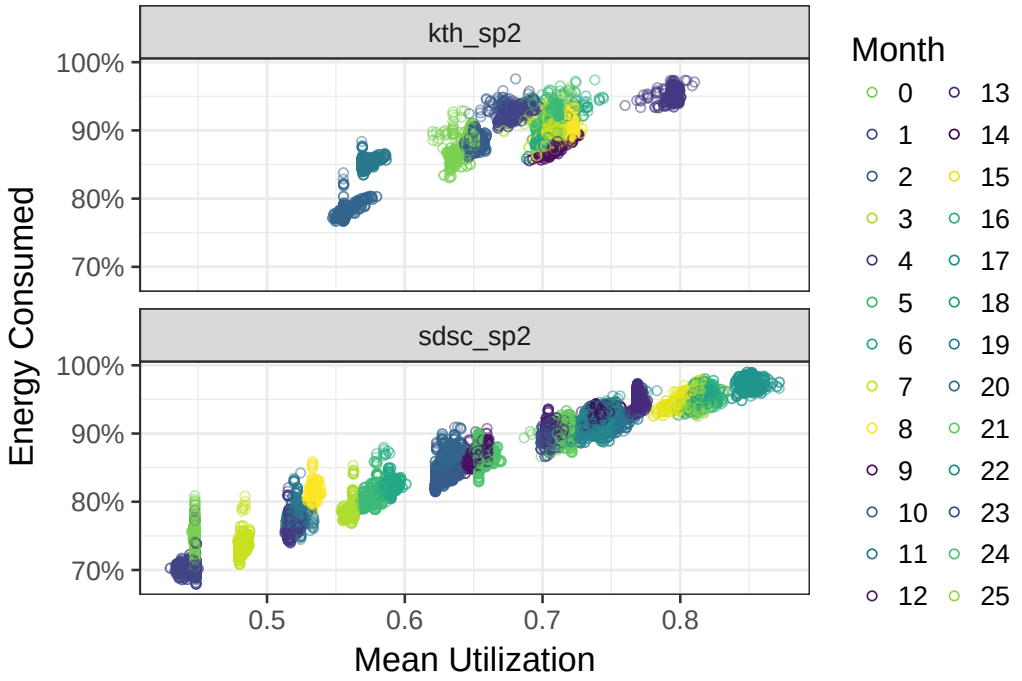


Figure 7.6: Energy saving opportunities fluctuate over time. All simulation results are shown but EASY (which is used as a reference). Months are colored separately to underline the variety of inputs used in our experiments.

7.6.5 Opportunistic Shutdown

The main parameter of the opportunistic shutdown technique is t_{idle}. As expected, decreasing t_{idle} globally decreases the energy consumption. It is clearly the case for SDSC SP2 as seen on figure 7.8.

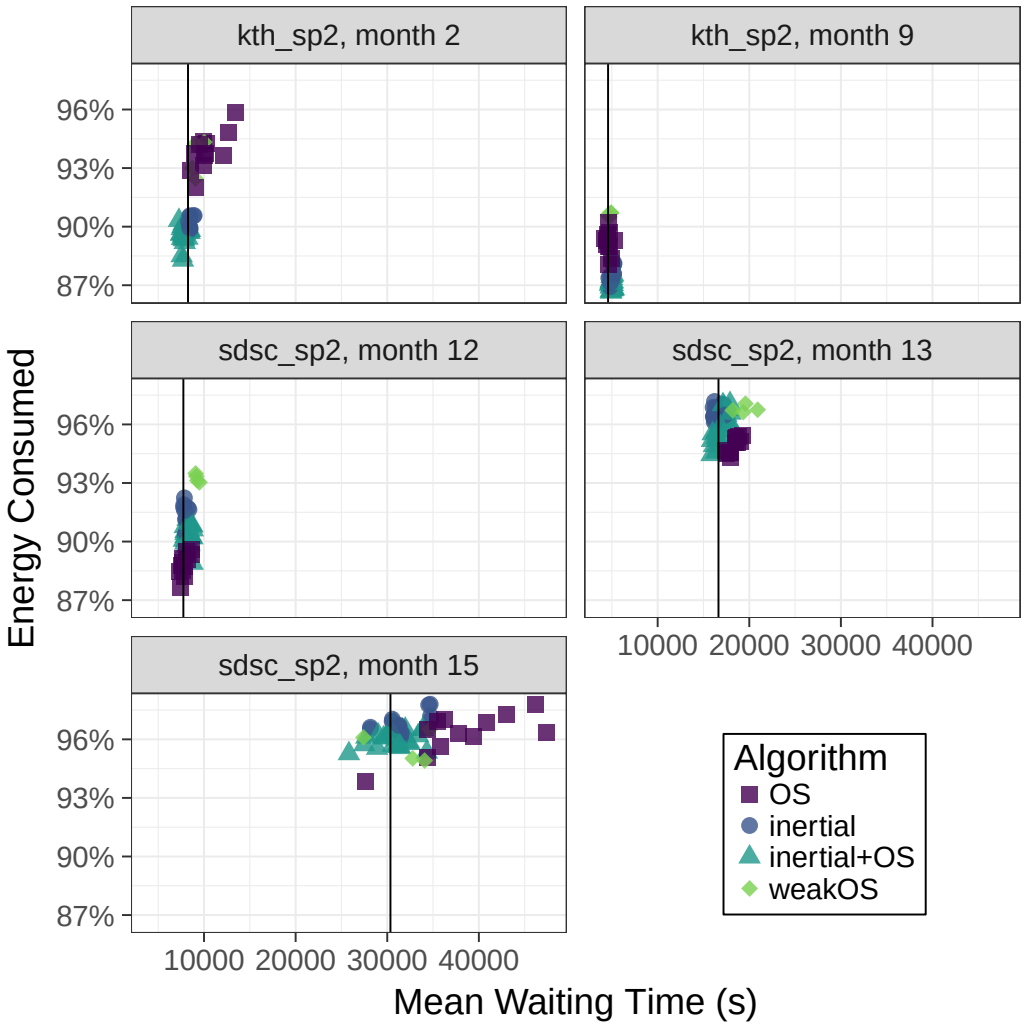


Figure 7.7: Figuration of energy and performance trade-offs for some months. The patterns exhibited in these examples are representative of most months in our experiment.

7.6.6 Inertial Shutdown

The main parameter of the inertial shutdown technique is $\bar{v}_{ub}$, as it evidences clear trade-offs between consumed energy and mean waiting time, as seen on figure 7.9 for the SDSC SP2 workload — results are similar on KTH SP2. Increasing $\bar{v}_{ub}$ slightly decreases the consumed energy and substantially increases the mean waiting time. This behaviour is expected, as this parameter has been introduced to control the mean waiting time increase. T is also an important parameter of the inertial shutdown technique — also seen on figure 7.9. Decreasing T directly decreases the energy consumption. The lower $\bar{v}_{ub}$, the clearer the impact of T on the consumed energy.

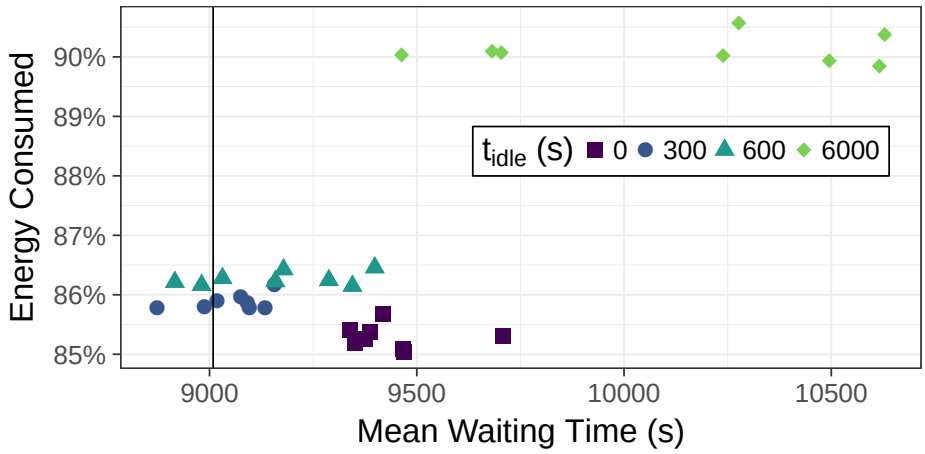


Figure 7.8: Impact of the t_{idle} parameter of the opportunistic shutdown technique (used alone) for the SDSC SP2 workload.

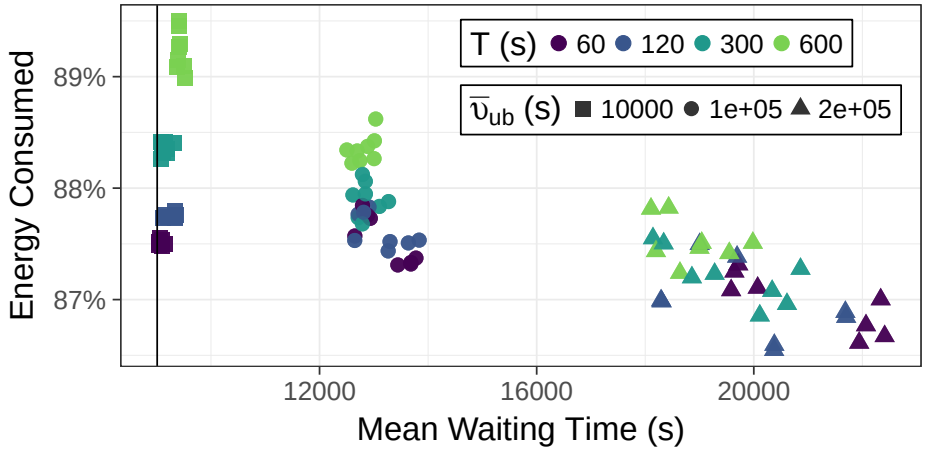


Figure 7.9: Impact of the $\bar{v}_{ub}$ and T parameters of the inertial shutdown technique (used alone) for the SDSC SP2 workload.

7.6.7 Inertial+Opportunistic Shutdown

The way we mixed both techniques led the *inertial+OS* instances to mostly follow the behaviour of the corresponding *inertial* instances. Thus, the main parameter to describe these instances is $\bar{v}_{ub}$, as figure 7.10 shows.

If we only consider low $\bar{v}_{ub}$ values (most interesting trade-offs, as seen in section 7.6.2), minimizing t_{idle} is interesting as it allows to reduce both the energy consumption and the mean waiting time, as figure 7.10 shows. However, please keep in mind that this combination of techniques may lead to a high number of power switches.

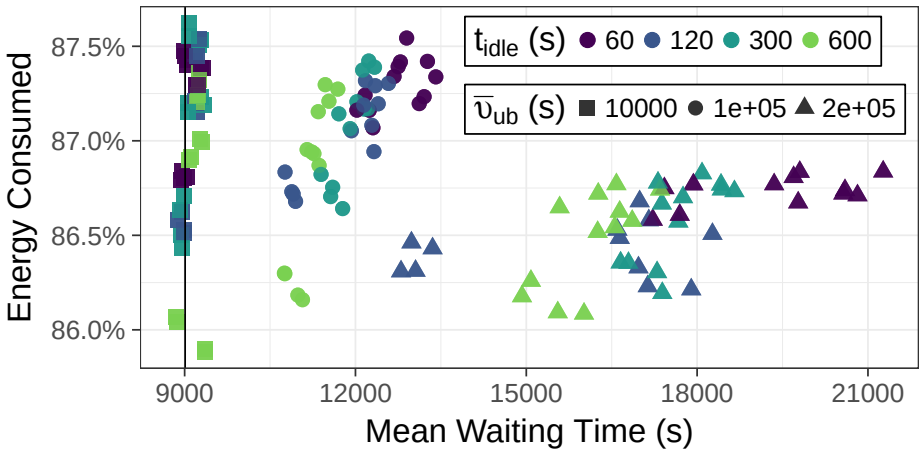


Figure 7.10: Impact of the $\bar{v}_{ub}$ and t_{idle} parameters when both inertial shutdown and opportunistic shutdown are used, for the SDSC SP2 workload.

7.7 Related Work

As we discussed in the introduction, the topic of energy saving in clusters received an increasing interest from the community. There are dedicated workshops and sessions in most conferences in the domain. However, to the best of our knowledge, there is no similar work directly comparable to ours that consider shutdown techniques to produce energy-performance trade-offs at the platform management level.

In particular, this subject was studied through various theoretical (idealized) models, see for instance the nice survey of ALBERS [Alb10], which analyzed competitive algorithms for both speed-scaling and shutdown techniques. Another direction is to model the system by Markov chains [HK12]. However, such methods do not lead to implementations.

On a more practical side, speed-scaling and shutdown techniques have been implemented in the Slurm resource manager and used to respect a power budget in [GGT15]. Modifications of EASY to respect an energy budget are conducted in [Dut+16a]. Finally, at the application level, ETINSKI et al. study energy-performance trade-offs for parallel applications in [Eti+12b].

Another view of the problem is to consider overprovisioning. SAROOD et al. maximize the throughput of moldable jobs under a strict power budget in [Sar+14] thanks to linear programs. PATKI et al. show in [Pat+16b] that hardware overprovisioned systems can be economically viable.

BENOIT et al. explore how much energy can be saved thanks to shutdown techniques in [Ben+17]. This article models many practical (physical) constraints that must be dealt with when large numbers of nodes are switched on and off. It evaluates how much energy can be saved in practice while respecting the different constraints. The approach is quite different to ours, as the job schedule is a fixed input. The potential gains under the different constraints are evaluated in a clairvoyant offline scenario, where the algorithms schedule off reservations with the objective of minimizing the energy consumption without moving the jobs. This approach cannot be used to compute an optimal solution of the problem studied in this chapter, as moving jobs may lead to more energy savings — but it may nevertheless be used as a *good* solution, as bad energy decisions are avoided.

The closest approach may be found in the general methodology proposed by ORGERIE et al. [OLG08], which aims at reducing energy via shutdown techniques in platforms that rely on advance reservations. In the proposed approach, the user selects a reservation between several choices that are made by the RJMS. When a resource becomes available, the RJMS selects whether it should be switched-off depending on the amount of time the resource is expected to remain idle. Predictions are done based on the recent platform usage to estimate when the next reservation will occur. Idle resources are used with a higher priority than off ones in order to minimize the number of switches — just as in the algorithms proposed in this chapter. Impressive results are given on the Grid'5000 workload — up to 30 % energy savings. However, these results are not really comparable to ours as the Grid'5000 workload greatly differs from classical HPC workloads — most jobs are reserved in advance and the platform utilization if very low. Evaluating how the proposed techniques behave in the context studied in this chapter — e.g., by creating advance reservations for the jobs submission times or without advance reservations and with the chaos produced by EASY backfilling and badly-estimated user-given execution times — would be an interesting future work.

7.8 Conclusion and Future Work

We have presented in this chapter two main techniques to produce energy and performance trade-offs. A parametric opportunistic shutdown technique has been used in combination with an off reservation technique, which led to energy gains up to 25 % with little performance loss.

While keeping an off reservation of almost fixed size does not seem auspicious, adapting this size to the system responsiveness is very promising. It indeed resulted in significant energy savings, reduced responsiveness losses, reasonable numbers of switches and an easily predictable behavior. These advantages make this technique a good candidate for implementations in production, as it would allow significant energy savings at low risk.

In this work, we have considered rigid jobs whose execution time is placement independent. As a future work we would like to take locality into account, as placing jobs locally reduces their execution times on most clusters and therefore their energy cost. We think that dramatic energy savings can be obtained by using locality constraints wisely. Finally, we think that using the same type of techniques on another basis than EASY could lead to higher energy savings — e.g., conservative backfilling [MF01] and its variants [Lin+13]. We indeed avoided quite strongly to hinder the priority job in our implementation and we think that lowering this constraint while avoiding starvation may lead to interesting results.

8 Conclusion

Need of a *new* simulator HPC platforms are complex distributed systems, and they are therefore very difficult to manage. Comparing the quality of resource management policies is arduous since each policy achieves a compromise between several conflicting objectives and potentially deals with different constraints. As energy consumption is a major limitation to build bigger platforms — in particular on the road to exascale — this work focused on methods to save energy. In this dissertation we took a holistic approach to study resource management strategies on HPC platforms rather than proposing yet another dedicated method to look into particular energy-related problems.

Our approach was to study how to manage HPC platforms by simulation as it allows to explore many scenarios at reasonable cost. It considers platforms as complex distributed systems and simulates them as such in the hope of improving the soundness of the results. To this end we emphasized separation of concerns in such a way that state-of-the-art distributed simulation frameworks could be employed internally. Separation of concerns is also stressed in our approach by the strong decoupling between the platform simulation and the decision making. First, this allows to study management policies as a whole and avoids to confine them to particular paradigms — e.g., queue-based job management is very popular but other methods exist [Zhe+16]. Second, this allows to assess and compare various implementations of resource managers — notably production RJMSs and academic prototypes — regardless of the programming language they use.

The roots of this methodology come from the growing awareness that experiments are generally insufficient regarding reproducibility and solidity in scientific experimental fields. We believe that increasing modularity in the simulation of such systems was a necessary step to improve further studies with less effort, especially towards larger scales.

This work resulted in both our methodology and its implementation in the Batsim simulator. Substantial effort has been made in this implementation as it is meant to last. For this purpose, Batsim has been and remains heavily tested, and its results have been evaluated against the real RJMS OAR. It is currently involved in several experiments conducted in different teams all over the world. Batsim is

general-purpose and can be used in manifold scenarios with various simulation models.

As a case study, we addressed the question of saving energy at the platform management level. To this end we first studied the problem of maximizing the system performances with a limited available amount of energy during a given time period. We proposed several extensions to EASY backfilling to face this problem. The main idea behind these algorithms is to conserve unused energy and to use it later on to execute jobs. Contrary to classical power capping algorithms, this mechanism can schedule jobs that request a lot of energy and thus leads to better performances. The energy is saved by either idling or switching-off machines that are not used. As we worked on the above-mentioned problem, we felt that trade-offs between performances and energy savings were worth further investigation. To that extent we studied more generally the problem as a multi-objective one. In this case we were interested in trade-off solutions that reduce the energy consumption without deteriorating the performances too much. This work showed that shutdown techniques can achieve significant energy savings by itself. The main result is that deliberately switching and keeping machines off in accordance with the current load pressure leads to better trade-offs than switching the machines off opportunistically — i.e., to shutdown machines if they remain idle. The proposed algorithm has also the advantages of being more predictable and to require many fewer switches than the opportunistic technique.

Perspectives As most research studies on complex systems, we are aware that our work has not addressed several aspects that deserve further attention. As evaluating the credibility of simulators is hard and complex [MLW01; Flo06], there remains room for improvements in the Batsim evaluation process. For instance, validations with more machines, more jobs and longer workloads would be interesting to consider but may involve unreasonable costs. Another path to consider is to compare Batsim results to RJMSs other than OAR. Conducting such experiments with Slurm [YJG03] seems knotty and costly [Luc11; TB15; Gon+17] but connecting Flux [Ahn+14] to Batsim looks promising.

More generally, Batsim does provide realistic results but their soundness is currently mostly left to the users and depends on a combination of multiple parameters — mainly about the computing platform, the jobs and the phenomena to consider. While a part of this responsibility is to be placed on the simulator users, we think that gaining more insights on the limits of the different models would be beneficial. In particular, many open questions remain about the modeling of applications and

deserve further study. Coarse-grained models are arguably the most pertinent at the platform management level as one generally seeks the big picture. However, determining the relevance of each job model depending on the phenomena one wants to observe — and to evaluate the trade-offs between realism and simulation speed granted by each one of them — is hard and merit future work. This is notably the case for temporal network congestion and I/O data movements that occur on (heterogeneous) platforms.

On the side of applying Batsim, many opportunities remain to increase energy savings at the platform management level. Our work essentially focused on shutdown techniques to save energy. We believe that correctly estimating — e.g., with learning techniques — which events are likely to occur at the next moments and the associated risks can reduce the number of *bad* shutdown decisions and therefore increase energy savings. Our work considered jobs as context-independent to focus on the impact on shutdown techniques in isolation. However, as the execution of most jobs is context-dependent, we think that considering data movements and network communications at this level is worth addressing as it may be a major leverage to reduce the jobs individual energy. Taking these phenomena realistically into consideration may however require extra studies on application modeling. Another way to reduce the individual energy of jobs is to allow DVFS decisions at the platform level. This is risky since such decisions can increase the jobs overall energy, but learning techniques can be used to estimate how the applications behave since users often submit the same applications [Wal+16].

List of Figures

List of Tables

Bibliography

[@bataar] oar-team. *Bataar on Github*. https://github.com/oar-team/oar3/blob/master/oar/kao/bataar.py. 2017. cit. on p. 34

[@batctn] Inria. *Batsimctn Project on the Inria Forge*. https://gforge.inria.fr/projects/simctn/. Feb. 2016. cit. on p. 39

[@batexpe] Inria. *expe_batsim Project on the Inria Forge*. https://gforge.inria.fr/projects/expe-batsim. Feb. 2016. cit. on p. 41

[@batgit1] Batsim Team. *Batsim Github Repository*. https://github.com/oar-team/batsim. 2017. cit. on pp. 4, 24, 30, 44

[@batgit2] Batsim Team. *Batsim Gitlab Repository*. https://gitlab.inria.fr/batsim/batsim. 2017. cit. on pp. 4, 24, 88

[@batproto] oar-team. *Batsim Protocol Description on Github*. https://github.com/oar-team/batsim/blob/master/doc/proto_description.md. 2017. cit. on pp. 26, 27

[@batsched] Millian Poquet. *Batsched Gitlab Repository*. https://gitlab.inria.fr/batsim/batsched. 2017. cit. on pp. 86, 88

[@evalys] Inria. *Evalys Project on Github*. https://github.com/oar-team/evalys. cit. on p. 41

[@execo] Inria. *Execo Project on the Inria Forge*. http://execo.gforge.inria.fr/doc/latest-stable/. Feb. 2016. cit. on p. 40

[@extrae] Barcelona Supercomputing Center. *Extrae*. https://www.bsc.es/computer-sciences/extrae. Feb. 2016. cit. on p. 35

[@extrea2tit] Lucas Mello Schnorr. *tcbozzetti/trabalhoconclusao*. https://github.com/schnorr/trabalhoconclusao/blob/master/LabBook.org. Feb. 2016. cit. on p. 35

[@green500] *GREEN500 wensite*. URL: https://www.top500.org/green500/ (visited on Sept. 26, 2017). cit. on pp. 5, 6

[@greenlap2017] *TOP500 Meanderings: Supercomputers Take Big Green Leap in 2017*. URL: https://www.top500.org/news/top500-meanderings-supercomputers-take-big-green-leap-in-2017/ (visited on Sept. 26, 2017). cit. on p. 5

[@kamelot] oar-team. *Kamelot on Github*. https://github.com/oar-team/oar3/blob/master/oar/kao/kamelot.py. 2017. cit. on p. 34

[@nancyG5K] Grid5000. *Grid5000 Nancy Clusters Description*. https://www.grid5000.fr/mediawiki/index.php/Nancy:Home. Feb. 2016. cit. on p. 34

[@ns3] *The ns-3 network simulator*. URL: https://www.nsnam.org (visited on Sept. 30, 2017). cit. on p. 2

[@pizdaint] *Piz Daint supercomputer description*. URL: https://www.top500.org/system/177824 (visited on Sept. 26, 2017). cit. on pp. 2, 5

[@pwa] Dror Feitelson. *Parallel Workload Archive*. http://www.cs.huji.ac.il/labs/parallel/workload/. 2017. cit. on p. 89

[@pybatsim] David Glesser, Millian Poquet, Henri Casanova, et al. *Pybatsim Gitlab Repository*. https://gitlab.inria.fr/batsim/pybatsim. 2017. cit. on p. 68

[@reprEB17] *Artifacts to reproduce the "Towards Energy Budget Control in HPC" article*. https://github.com/glesserd/energybudget-expe. 2017. cit. on p. 68

[@reprETO17] Pierre-François Dutot, Millian Poquet, and Denis Trystram. *Artifacts to reproduce the "Performance vs Energy Tradeoffs via Shutdown Policies in EASY Backfilling" article*. 2017. cit. on p. 87

[@sunwaytl] *Sunway Taihulight supercomputer description*. URL: https://www.top500.org/system/178764 (visited on Sept. 26, 2017). cit. on pp. 5, 57

[@tianhe2] *Tianhe-2 supercomputer description*. URL: https://www.top500.org/system/177999 (visited on Sept. 26, 2017). cit. on p. 5

[@top500] *TOP500 wensite*. URL: https://www.top500.org/ (visited on Sept. 30, 2017). cit. on p. 1

[Ahn+14] Dong H Ahn, Jim Garlick, Mark Grondona, et al. „Flux: A next-generation resource management framework for large hpc centers“. In: *Parallel Processing Workshops (ICCPW), 2014 43rd International Conference on*. IEEE. 2014, pp. 9–17. cit. on pp. 3, 24, 102

[Alb10] Susanne Albers. „Energy-efficient algorithms“. In: *Communications of the ACM* 53.5 (2010), pp. 86–96. cit. on pp. 81, 98

[Ash+10] Steve Ashby, Pete Beckman, Jackie Chen, et al. *Opportunities and Challenges of Exascale Computing*. Tech. rep. U.S. Department of Energy, 2010. cit. on p. 2

[Bal+12] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, et al. „Adding Virtualization Capabilities to the Grid'5000 Testbed“. In: *Cloud Computing and Services Science*. Communications in Computer and Information Science 367. DOI: 10.1007/978-3-319-04519-1_1. Springer International Publishing, Apr. 2012, pp. 3–20. cit. on pp. 33, 68

[Bam+16] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. „Energy-Aware Scheduling for Real-Time Systems: A Survey“. In: *ACM Transactions on Embedded Computing Systems (TECS)* 15.1 (2016), p. 7. cit. on p. 61

[Bat+15] Natalie Bates, Girish Ghatikar, Ghaleb Abdulla, et al. „Electrical Grid and Supercomputing Centers: An Investigative Analysis of Emerging Opportunities and Challenges“. In: *Informatik-Spektrum* 38.2 (2015), pp. 111–127. cit. on p. 57

[Ben+17] Anne Benoit, Laurent Lefèvre, Anne-Cécile Orgerie, and Issam Rais. „Reducing the energy consumption of large scale computing systems through combined shutdown policies with multiple constraints“. In: *International Journal of High Performance Computing Applications* (2017). cit. on p. 99

[BH07] Luiz André Barroso and Urs Hölzle. „The case for energy-proportional computing“. In: *Computer* 40.12 (2007). cit. on p. 61

[Bur+16] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. „Borg, omega, and kubernetes“. In: *Communications of the ACM* 59.5 (2016), pp. 50–57. cit. on p. 3

[Cap+05] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, et al. „A batch scheduler with high level components“. In: *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*. Vol. 2. IEEE. 2005, pp. 776–783. cit. on pp. 24, 33

[Cas+14] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. „Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms“. In: *Journal of Parallel and Distributed Computing* 74.10 (June 2014), pp. 2899–2917. cit. on pp. 4, 24, 27, 47, 48

[CG09] Yves Caniou and Jean-Sébastien Gay. „Simbatch: An API for simulating and predicting the performance of parallel resources managed by batch systems“. In: *Euro-Par 2008 Workshops-Parallel Processing*. Springer. 2009, pp. 223–234. cit. on p. 24

[Cha+01] Jeffrey S Chase, Darrell C Anderson, Prachi N Thakar, Amin M Vahdat, and Ronald P Doyle. „Managing energy and server resources in hosting centers“. In: *ACM SIGOPS operating systems review* 35.5 (2001), pp. 103–116. cit. on p. 2

[CM10] Sangyeun Cho and Rami G Melhem. „On the interplay of parallelization, program performance, and energy consumption“. In: *IEEE Transactions on Parallel and Distributed Systems* 21.3 (2010), pp. 342–353. cit. on p. 81

[Don+11] Jack Dongarra, Pete Beckman, Terry Moore, et al. „The international exascale software project roadmap“. In: *International Journal of High Performance Computing Applications* 25.1 (2011), pp. 3–60. cit. on pp. 1, 57

[Dut+09] Pierre-Francois Dutot, Krzysztof Rzadca, Erik Saule, Denis Trystram, et al. „Multi-objective scheduling“. In: *Introduction to scheduling* (2009), pp. 219–251. cit. on p. 80

[Dut+16a] Pierre-François Dutot, Yiannis Georgiou, David Glesser, et al. „Towards Energy Budget Control in HPC“. In: *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*. IEEE. 2016. cit. on p. 98

[Dut+16b] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. „Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator“. In: *Job Scheduling Strategies for Parallel Processing (JSSPP)*. 2016. cit. on p. 40

[Ell+17] D Ellsworth, T Patki, M Schulz, B Rountree, and A Malony. *Simulating Power Scheduling at Scale*. Tech. rep. Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2017. cit. on p. 2

[Eti+12a] Maja Etinski, Julita Corbalan, Jesus Labarta, and Mateo Valero. „Parallel job scheduling for power constrained HPC systems“. In: *Parallel Computing* 38.12 (2012), pp. 615–630. cit. on p. 60

[Eti+12b] Maja Etinski, Julita Corbalán, Jesús Labarta, and Mateo Valero. „Understanding the future of energy-performance trade-off via DVFS in HPC environments“. In: *Journal of Parallel and Distributed Computing* 72.4 (2012), pp. 579–590. cit. on p. 98

[Fei01] Dror G Feitelson. „Metrics for parallel job scheduling and their convergence“. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2001, pp. 188–205. cit. on pp. 12, 60

[Fei15] Dror G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2015. cit. on pp. 10, 35, 79

[Fei16] Dror G Feitelson. „Resampling with Feedback—A New Paradigm of Using Workload Data for Performance Evaluation“. In: *European Conference on Parallel Processing*. Springer. 2016, pp. 3–21. cit. on p. 44

[FF05] Eitan Frachtenberg and Dror G Feitelson. „Pitfalls in parallel job scheduling evaluation“. In: *Job Scheduling Strategies for Parallel Processing (JSSPP)*. Vol. 3834. Springer. 2005, pp. 257–282. cit. on p. 60

[Flo06] Sally Floyd. „Maintaining a critical attitude towards simulation results (invited talk)“. In: (2006). cit. on pp. 2, 102

[FTK14] Dror G Feitelson, Dan Tsafrir, and David Krakov. „Experience with using the parallel workloads archive“. In: *Journal of Parallel and Distributed Computing* 74.10 (2014), pp. 2967–2982. cit. on pp. 69, 88

[Geo+14] Yiannis Georgiou, Thomas Cadeau, David Glesser, et al. „Energy Accounting and Control with SLURM Resource and Job Management System.“ In: *International Conference on Distributed Computing and Networking (ICDCN)* 8314 (2014), pp. 96–118. cit. on p. 59

[Geo+15] Yiannis Georgiou, David Glesser, Krzysztof Rzadca, and Denis Trystram. „A Scheduler-Level Incentive Mechanism for Energy Efficiency in HPC“. In: *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE. 2015, pp. 617–626. cit. on pp. 2, 59

[GGT15] Yiannis Georgiou, David Glesser, and Denis Trystram. „Adaptive Resource and Job Management for Limited Power Consumption“. In: *IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPS 2015, Hyderabad, India, May 25-29*. Hyderabad, India, 2015, pp. 863–870. cit. on pp. 59, 61, 81, 98

[GK07] Rodolphe Giroudeau and Jean-Claude König. „Scheduling with Communication Delay“. In: *Multiprocessor Scheduling: Theory and Applications*. ARS Publishing, Dec. 2007, pp. 1–26. cit. on p. 46

[Gon+17] P Gonzalo, Erik Elmroth, PO Östberg, Lavanya Ramakrishnan, et al. „ScSF: a scheduling simulation framework“. In: *21th Workshop on Job Scheduling Strategies for Parallel Processing (JSSP 2017), Orlando FL, USA, June 2nd, 2017*. 2017. cit. on p. 102

[HCS11] Sascha Hunold, Henri Casanova, and Frédéric Suter. „From simulation to experiment: a case study on multiprocessor task scheduling“. In: *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*. IEEE. 2011, pp. 665–672. cit. on p. 47

[HHN08] Junichi Hikita, Akio Hirano, and Hiroshi Nakashima. „Saving 200kw and $200 k/year by power-aware job/machine scheduling“. In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE. 2008, pp. 1–8. cit. on p. 61

[Hin+11] Benjamin Hindman, Andy Konwinski, Matei Zaharia, et al. „Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.“ In: *NSDI*. Vol. 11. 2011. 2011, pp. 22–22. cit. on p. 3

[Hin13] Pieter Hintjens. *ZeroMQ: messaging for many applications*. O'Reilly Media, Inc., 2013. cit. on p. 26

[HK12] Matthias Herlich and Holger Karl. „Average and Competitive Analysis of Latency and Power Consumption of a Queuing System with a Sleep Mode“. In: *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*. e-Energy '12. Madrid, Spain: ACM, 2012, 14:1–14:10. cit. on p. 98

[Hun15] Sascha Hunold. „One Step towards Bridging the Gap between Theory and Practice in Moldable Task Scheduling with Precedence Constraints“. In: *Concurrency and Computation: Practice and Experience* 27.4 (2015), pp. 1010–1026. cit. on p. 46

[Jea+13] Emmanuel Jeannot, Esteban Meneses, Guillaume Mercier, François Tessier, and Gengbin Zheng. „Communication and topology-aware load balancing in charm++ with treematch“. In: *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1–8. cit. on p. 47

[Khe+14] Bhavesh Khemka, Ryan Friese, Sudeep Pasricha, et al. „Utility Driven Dynamic Resource Management in an Oversubscribed Energy-Constrained Heterogeneous System“. In: *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE. 2014, pp. 58–67. cit. on p. 61

[KR10] Dalibor Klusáček and Hana Rudová. „Alea 2: job scheduling simulator“. In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2010, p. 61. cit. on pp. 2, 23

[Leu04] J.Y.T. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC Computer and Information Science Series. CRC Press, 2004. cit. on p. 46

[Lif95] David A. Lifka. „The ANL/IBM SP Scheduling System“. In: *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. IPPS '95. London, UK, UK: Springer-Verlag, 1995, pp. 295–303. cit. on p. 47

[Lin+13] Alexander M Lindsay, Maxwell Galloway-Carson, Christopher R Johnson, David P Bunde, and Vitus J Leung. „Backfilling with guarantees made as jobs arrive“. In: *Concurrency and Computation: Practice and Experience* 25.4 (2013), pp. 513–523. cit. on p. 100

[Liu+15] Ning Liu, Xi Yang, Xian-He Sun, Johnathan Jenkins, and Robert Ross. „Yarnsim: Simulating hadoop yarn“. In: *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE. 2015, pp. 637–646. cit. on p. 3

[Luc+14] Robert Lucas, James Ang, Keren Bergman, et al. *DOE Advanced Scientific Computing Advisory Subcommittee (ASCAC) Report: Top Ten Exascale Research Challenges*. Feb. 2014. cit. on p. 1

[Luc+15] Giorgio Lucarelli, Fernando Mendonca, Denis Trystram, and Frederic Wagner. „Contiguity and Locality in Backfilling Scheduling“. In: *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 586–595. cit. on pp. 49, 50, 51, 59

[Luc11] Alejandro Lucero. „Simulation of batch scheduling using real production-ready software tools“. In: *Proceedings of the 5th IBERGRID* (2011). cit. on pp. 24, 102

[Mer16] Michael Mercier. *MPI+PRV+TIT-traces_NAS-Benchmarks_2016-02-08–10-10-44*. `http://academictorrents.com/details/53b46a4ff43a8ae91f674b26c65c5cc6` Feb. 2016. cit. on p. 41

[MF01] Ahuva W. Mu'alem and Dror G. Feitelson. „Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling“. In: *Parallel and Distributed Systems, IEEE Transactions on* 12.6 (2001), pp. 529–543. cit. on pp. 7, 43, 47, 50, 58, 62, 83, 100

[MLW01] Paul R Muessig, Dennis R Laack, and John J Wrobleski. *An integrated approach to evaluating simulation credibility*. Tech. rep. NAVAL AIR WARFARE CENTER WEAPONS DIV CHINA LAKE CA, 2001. cit. on p. 102

[Mor08] Richard D Morey. „Confidence intervals from normalized data: A correction to Cousineau (2005)“. In: *Reason* 4.2 (2008), pp. 61–64. cit. on p. 70

[MV15] Prakash Murali and Sathish Vadhiyar. „Metascheduling of HPC Jobs in Day-Ahead Electricity Markets“. In: *High Performance Computing (HiPC), 2015 IEEE 22nd International Conference on*. IEEE. 2015, pp. 386–395. cit. on p. 61

[NAS16] NASA. *NAS Parallel Benchmarks*. `https://www.nas.nasa.gov/publications/npb.html`. Feb. 2016. cit. on p. 34

[Ngo+16] Yanik Ngoko, Denis Trystram, Valentin Reis, and Christophe Cérin. „An Automatic Tuning System for Solving NP-Hard Problems in Clouds“. In: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2016, Chicago, IL, USA, May 23-27, 2016*. 2016, pp. 1443–1452. cit. on p. 2

[NS17] Tchimou N'takpé and Frédéric Suter. „Don't Hurry be Happy: a Deadline-based Backfilling Approach“. In: *Job Scheduling Strategies for Parallel Processing (JSSPP)*. 2017. cit. on p. 4

[OLG08] Anne-Cécile Orgerie, Laurent Lefèvre, and Jean-Patrick Gelas. „Save Watts in your Grid: Green Strategies for Energy-Aware Framework in Large Scale Distributed Systems“. In: *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. Melbourne, Australia, Dec. 2008, pp. 171–178. cit. on pp. 5, 61, 99

[Pat+15] Tapasya Patki, David K Lowenthal, Anjana Sasidharan, et al. „Practical Resource Management in Power-Constrained, High Performance Computing“. In: *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM. 2015, pp. 121–132. cit. on p. 60

[Pat+16a] Tapasya Patki, Natalie Bates, Girish Ghatikar, et al. „Supercomputing Centers and Electricity Service Providers: A Geographically Distributed Perspective on Demand Management in Europe and the United States“. In: *International Conference on High Performance Computing*. Springer. 2016, pp. 243–260. cit. on pp. 57, 76

[Pat+16b] Tapasya Patki, David K Lowenthal, Barry L Rountree, Martin Schulz, and Bronis R de Supinski. „Economic viability of hardware overprovisioning in power-constrained high performance computing“. In: *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*. IEEE Press. 2016, pp. 8–15. cit. on p. 98

[PM05] Fco Javier Ridruejo Perez and José Miguel-Alonso. „INSEE: An interconnection network simulation and evaluation environment“. In: *Euro-Par 2005 Parallel Processing*. Springer, 2005, pp. 1014–1023. cit. on pp. 2, 24

[PML15] Jose A Pascual, Jose Miguel-Alonso, and Jose A Lozano. „Locality-aware policies to improve job scheduling on 3D tori“. In: *The Journal of Supercomputing* 71.3 (2015), pp. 966–994. cit. on p. 24

[PNM09] Jose Antonio Pascual, Javier Navaridas, and José Miguel-Alonso. „Effects of Topology-Aware Allocation Policies on Scheduling Performance“. In: *Job Scheduling Strategies for Parallel Processing, 14th International Workshop, JSSPP 2009, Rome, Italy, May 29, 2009. Revised Papers*. 2009, pp. 138–156. cit. on p. 47

[Rou+12] Barry Rountree, Dong H Ahn, Bronis R De Supinski, David K Lowenthal, and Martin Schulz. „Beyond DVFS: A first look at performance under a hardware-enforced power bound“. In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE. 2012, pp. 947–953. cit. on p. 60

[Rui+15] Cristian Ruiz, Salem Harrache, Michael Mercier, and Olivier Richard. „Reconstructable Software Appliances with Kameleon“. In: *SIGOPS Oper. Syst. Rev.* 49.1 (Jan. 2015), pp. 80–89. cit. on pp. 39, 87

[Sar+14] Osman Sarood, Akhil Langer, Abhishek Gupta, and Laxmikant Kale. „Maximizing throughput of overprovisioned hpc data centers under a strict power budget“. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press. 2014, pp. 807–818. cit. on pp. 60, 98

[Sin07] Oliver Sinnen. *Task Scheduling for Parallel Systems*. Wiley Series on Parallel and Distributed Computing. Wiley, 2007. cit. on p. 46

[SRH05] David C Snowdon, Sergio Ruocco, and Gernot Heiser. „Power management and dynamic voltage scaling: Myths and facts“. In: *Proceedings of the 2005 workshop on power aware real-time computing*. Vol. 12. 2005. cit. on pp. 7, 81

[SSE06] Oliver Sinnen, Leonel Augusto Sousa, and Frode Eika Sandnes. „Toward a Realistic Task Scheduling Model“. In: *IEEE Trans. Parallel Distrib. Syst.* 17.3 (Mar. 2006), pp. 263–275. cit. on p. 46

[TB15] Stephen Trofinoff and Massimo Benini. „Using and Modifying the BSC Slurm Workload Simulator“. In: (2015). cit. on p. 102

[Vav+13] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, et al. „Apache hadoop yarn: Yet another resource negotiator“. In: *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM. 2013, p. 5. cit. on p. 3

[Wal+16] Sean Wallace, Xu Yang, William E Vishwanath Venkatram andu Allcock, et al. „A data driven scheduling approach for power management on HPC systems“. In: *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE. 2016, pp. 656–666. cit. on p. 103

[Yan+13] Xu Yang, Zhou Zhou, Sean Wallace, et al. „Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems“. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM. 2013, p. 60. cit. on p. 61

[YB05] Jia Yu and Rajkumar Buyya. „A Taxonomy of Scientific Workflow Systems for Grid Computing“. In: *SIGMOD Rec.* 34.3 (Sept. 2005), pp. 44–49. cit. on p. 40

[YJG03] Andy B Yoo, Morris A Jette, and Mark Grondona. „Slurm: Simple linux utility for resource management“. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2003, pp. 44–60. cit. on pp. 24, 102

[Zhe+16] Xingwu Zheng, Zhou Zhou, Xu Yang, Zhiling Lan, and Jia Wang. „Exploring Plan-Based Scheduling for Large-Scale Computing Systems“. In: *Cluster Computing (CLUSTER), 2016 IEEE International Conference on*. IEEE. 2016, pp. 259–268. cit. on p. 101

Additionally, the work conducted in this dissertation directly led to the following communications.

Peer-reviewed international conferences

- Pierre-François Dutot, Yiannis Georgiou, David Glesser, et al. „Towards Energy Budget Control in HPC“. In: *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*. IEEE. 2016.

Peer-reviewed international workshops

- Pierre-François Dutot, Millian Poquet, and Denis Trystram. „Communication models insights meet simulations“. In: *International European Conference on Parallel and Distributed Computing (Euro-Par) Workshops*. 2015.
- Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. „Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator“. In: *Job Scheduling Strategies for Parallel Processing (JSSPP)*. 2016.

National workshops

- Pierre-François Dutot, Millian Poquet, and Denis Trystram. „Energy and Responsiveness Trade-offs in EASY Backfilling“. In: *Green Days@Sophia*. 2017.

Abstract

Computing platforms increasingly grow in power and complexity. Numerous challenges remain to build next generations of platforms, but exploiting the platforms is a challenge per se. Constraints such as energy consumption, data movements and resilience risk to initiate breaking points in the way that the platforms are managed — especially with the convergence of the different types of distributed platforms.

Resource and Jobs Management Systems (RJMSs) are critical middlewares that allow users to exploit the resources of such platforms. They must evolve to make the best use of the computing platforms while complying with these new constraints. Each evolution ideally require many iterations, but conducting them *in vivo* is not reasonable due to huge overhead. Simulation is an efficient way to tackle the subsequent problems, but particular caution must be taken when drawing results from simulation as using ill-suited models may lead to invalid results.

The first contribution of this dissertation is the proposition of a modular simulation methodology to study RJMSs and their evolution realistically — and the related simulator Batsim. The main idea is to strongly separate the simulation from the decision-making algorithms. This allows separation of concerns as any algorithm can benefit from a validated simulation with multiple levels of realism (features, accuracy of the models). This methodology improves the production launch of new policies since both academic prototypes and production RJMSs can be studied in the same context.

Batsim is used in the second part of this dissertation, which focuses on online and non-clairvoyant resource management policies to save energy. Several algorithms are first proposed and analyzed to maximize performances under an energy budget for a given time period. This dissertation then explores more generally possible energy and performances trade-offs that can be obtained with node shutdown techniques.

Résumé

Les plateformes de calcul se multiplient, grandissent en taille et gagnent en complexité. De nombreux défis restent à relever pour construire les prochaines générations de plateformes, mais exploiter cesdites plateformes est également un défi en soi. Des contraintes comme la consommation énergétique, les mouvements de données ou la résilience risquent de devenir prépondérantes et de s'ajouter à la complexité actuelle de la gestion des plateformes. Les méthodes de gestion de ressources peuvent également évoluer avec la convergence des différents types de plateformes distribuées.

Les gestionnaires de ressources sont des systèmes critiques au cœur des plateformes qui permettent aux utilisateurs d'exploiter les ressources. Les faire évoluer est nécessaire pour exploiter au mieux les ressources en prenant en compte ces nouvelles contraintes. Ce processus d'évolution est risqué et nécessite de nombreuses itérations qu'il semble peu raisonnable de réaliser *in vivo* tant les coûts impliqués sont importants. La simulation, beaucoup moins coûteuse, est généralement préférée pour faire ce type d'études mais pose des questions quant au réalisme des résultats ainsi obtenus.

La première contribution de cette thèse est de proposer une méthode de simulation modulaire pour étudier les gestionnaires de ressources et leur évolution — ainsi que le simulateur résultant nommé Batsim. L'idée principale est de séparer fortement la simulation et les algorithmes de prise de décision. Cela permet une séparation des préoccupations puisque les algorithmes, quels qu'ils soient, peuvent bénéficier d'une simulation validée proposant différents niveaux de réalisme. Cette méthode simplifie la mise en production de nouvelles politiques puisque des codes issus à la fois de gestionnaires de ressources de production et de prototypes académiques peuvent être étudiés dans le même contexte.

La méthode de simulation proposée est illustrée dans la seconde partie de cette thèse, qui s'intéresse à des problèmes de gestion de ressources non clairvoyants mêlant optimisation des performances et de la consommation énergétique. Différents algorithmes sont d'abord proposés et étudiés afin de respecter un budget d'énergie pendant une période de temps donnée. Nous étudions ensuite plus généralement les différents compromis réalisables entre performances et énergie grâce à différentes politiques d'extinction de nœuds de calcul.