Road to exascale: Improving scheduling performances and reducing energy consumption with the help of end-users

David Glesser

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE la Communauté UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 7 Août 2006

Présentée par

David Glesser

Thèse dirigée par **Denis Trystram**

préparée au sein du **Laboratoire Informatique de Grenoble**
et de **Ecole Doctorale de Mathématiques, Sciences et Technologies de l'Information, Informatique**

Road to exascale: Improving scheduling performances and reducing energy consumption with the help of end-users

Thèse soutenue publiquement le **18 octobre 2016**,
devant le jury composé de :

Patrick Gros
Directeur de Recherche, Inria, France, Président

Walfredo Cirne
Chercheur senior, Google, USA, Rapporteur

Jean-Marc Pierson
Professeur d'université, IRIT Laboratory, Université Paul Sabatier Toulouse 3, France, Rapporteur

Yiannis Georgiou
Architecte logiciel, Bull Atos Technologies, France, Examinateur

Jean-Marc Nicod
Professeur d'université, AS2M/FEMTO-ST Lab, Université de Franche-Comté, France, Examinateur

Denis Trystram
Professeur d'université, Grenoble Institute of Technology, Université Grenoble Alpes, France, Directeur de thèse

Abstract

The field of High Performance Computing (HPC) is characterized by the continuous evolution of computing architectures, the proliferation of computing resources and the increasing complexity of applications users wish to solve. One of the most important software of the HPC stack is the Resource and Job Management System (RJMS) which stands between the user workloads and the platform, the applications and the resources. This specialized software provides functions for building, submitting, scheduling and monitoring jobs in a dynamic and complex computing environment.

In order to reach exaflops HPC systems, new constraints and objectives have been introduced. This thesis develops and tests the idea that the users of such systems can help reaching the exaflopic scale. Specifically, we show and introduce new techniques that employ users behaviors to improve energy consumption and overall cluster performances.

To test the proposed techniques, we need to develop new tools and methodologies that scale up to large HPC clusters. Thus, we designed adequate tools that assess new RJMS scheduling algorithms of such large systems. These tools are able to run on small clusters by emulating or simulating bigger platforms. After evaluating different techniques to measure the energy consumption of HPC clusters, we propose a new heuristic, based on the popular Easy Backfilling algorithm, in order to control the power consumption of such huge systems. We also demonstrate, using the same idea, how to control the energy consumption during a time period. The proposed mechanism is able to limit the energy consumption while keeping satisfying performances. If energy is a limited resource, it has to be shared fairly. We also present a mechanism which shares energy consumption among users. We argue that sharing fairly the energy among users should motivate them to reduce the energy consumption of their applications. Finally, we analyze past and present behaviors of users using learning algorithms in order to improve the performances of the parallel platforms. This approach does not only outperform state of the art methods, it also shows promising insight on how such method can improve other aspects of RJMS.

Résumé

Le domaine du calcul haute performance (*i.e.* la science des supercalculateurs) est caractérisé par l'évolution continuelle des architectures de calcul, la prolifération des ressources de calcul et la complexité croissante des problèmes que les utilisateurs veulent résoudre. Un des logiciels les plus importants de la pile logicielle des supercalculateurs est le Système de Gestion des Ressources et des Tâches. Il est le lien entre la charge de travail donnée par les utilisateurs et la plateforme de calcul. Ce type de logiciels spécialisés fournit des fonctions pour construire, soumettre, planifier et surveiller les tâches de calculs dans un environnent complexe et dynamique.

Pour pouvoir atteindre des supercalculateurs exaflopiques, de nouvelles contraintes et objectifs ont été inventés. Cette thèse développe et teste l'idée que les utilisateurs de ces systèmes peuvent aider à atteindre l'échelle exaflopique. Spécifiquement, nous montrons des techniques qui utilisent les comportements des utilisateurs pour améliorer la consommation énergétique et les performances globales des supercalculateurs.

Pour tester ces nouvelles techniques, nous avons besoin de nouveaux outils et méthodes capables d'aller jusqu'à l'échelle exaflopique. Nous proposons donc des outils qui permettent de tester de nouveaux algorithmes capables de s'exécuter sur ces systèmes. Ces outils sont capables de fonctionner sur de petits supercalculateurs en émulant ou simulant des systèmes plus puissants. Après avoir évalué différentes techniques pour mesurer l'énergie dans les supercalculateurs, nous proposons une nouvelle heuristique, basée sur un algorithme répandu (Easy Backfilling), pour pouvoir contrôler la puissance électrique de ces énormes systèmes. Nous montrons aussi comment, en utilisant une méthode semblable, contrôler la consommation énergétique pendant une période de temps. Le mécanisme proposé peut limiter la consommation énergétique tout en gardant des performances satisfaisantes. Si l'énergie est une ressource limitée, il faut la partager équitablement. Nous présentons de plus un mécanisme permettant de partager la consommation énergétique entre les utilisateurs. Nous soutenons que cette méthode va motiver les utilisateurs à réduire la consommation énergétique de leurs calculs. Finalement, grâce à un algorithme d'apprentissage automatique, nous analysons le comportement actuel et passé des utilisateurs pour améliorer les performances des supercalculateurs. Cette approche non seulement surpasse les performances des travaux existants, mais aussi ouvre la voie à l'utilisation de méthodes semblables dans d'autres aspects des Systèmes de Gestion des Ressources et des Tâches.

Acknowledgement

First, I would like to thank the members of the jury for accepting to be the part of this committee.

A requirement for a good thesis is a good environment. So I would like to thank people who have made the $environments$ of my thesis wonderful.

Present and past members of MESCAL and MOAIS teams have created the ideal environment to create, debate, and laugh about research. I would like to wholeheartedly thank all of them. I especially thank my Ph.D. advisor Denis Trystram for the great and deep discussions even if we do not work in the same world.

Bull R&D is the second wonderful environment where my thesis work was done. I would like to thank Andry's teams to integrate me so fast into this huge company. Of course, I want to express my gratitude to Yiannis Georgiou for all the exchanges and time.

I would like to thank my friends who supported me and make me forbid for an evening or a weekend about my thesis.

My final thanks go to my family who has the difficult tasks to bear with me through these years.

Contents

List of Figures

List of Tables

1 Introduction

> *The last question was asked for the first time, half in jest, on May 21, 2061, at a time when humanity first stepped into the light. The question came about as a result of a five dollar bet over highballs, and it happened this way...*
>
> **— Isaac Asimov, The Last Question**

High performance computing (HPC) centers are huge complex machines built to process big computations. These computations allow scientists to simulate large physic simulations, like simulating the effect of a drug on HIV at the atom level. They can also be used to perform another kind of large calculations, like gene comparisons or statistics on the massive amount of data. We generally call "jobs" the computing software run on such HPC systems. Such systems are among the most sophisticated systems in term of number of components. Thousands of processors, made of billions of transistors, communicate together through complex networks in order to compute large simulations.

Managing this level of complexity is not an easy task. The biggest supercomputer as of June 2016 top500 list, Sunway TaihuLight, is composed of millions of cores[1], to reach the computational power of 93 petaflops. The current goal of the research community is to build exaflopic systems [Don+11]: billions of billions of operations per second executed concurrently on one machine. Even for smaller systems, making a large amount of computing components collaborate requires a high level of expertise. For one job to perform well on these systems a lot of engineering work is needed.

The complexity is not only on the components themselves. The facility around supercomputers needs to support a high energy consumption. For example, Sunway TaihuLight consumes around 15MW. Exaflopic systems, with today's technology, would consume power equivalent of a city of 200 000 inhabitants.

An opportunity to optimize the energy consumption and the performances is that such systems are not dedicated to one user running one job. Most of the time, a supercomputer runs multiple jobs on various dedicated computing components. In order to optimize these objectives some questions arise such as: *how to share these resources?*

This is the role of the software component called the Resource and Job Management System (RJMS). This middleware has the responsibility to choose where and when starting each job. "where" refers to which components are the most suitable to

[1] `https://www.top500.org/system/178764`

run this simulation. “when” refers to when these components will not be used by another job but also to some accounting policy (*e.g.* did the user pay to use these resources?).

Both of these scheduling questions (when and where) are answered with the goal of optimizing some objectives. We choose to classify performance measures in three categories. First, “administrative” objectives are related to the overall performance of the cluster. Such objectives interest mostly the owners or operators of supercomputers who want global metrics to measure the cost-effectiveness of their system. Second, “job” objectives aim at measuring the performance of the job on this system. Generally, these objectives are related to the completion time of the jobs. Finally, “user” objectives intent to measure how satisfied the end-user is with the system. The most popular measure, called bounded slowdown, is how fast the users obtain the result of their computations from the moment they have submitted their job.

Most existing approaches take the point of view of administrators. These algorithms and heuristics make the assumption of a perfect knowledge of the machine and then try to optimize administrative objectives. This approach fits well with existing approaches coming from the scheduling field. However, a big part of the problem is forgotten: it is users that deal with them.

This manner of tackling RJMS scheduling problems is not taken by laziness. On the first side, RJMS problems are mainly seen by system administrators and then presented to researchers with their point of view. On the other side, RJMS problems are hidden from users. They only observe how fast they obtain their results. Nevertheless, users have a different view on the system. They know well the jobs they launch. As a consequence, we believe that we should take advantages of both user and administrator knowledge in order to obtain the best possible schedule.

To successfully achieve an exaflopic cluster, everything should be optimized. From the hardware to the jobs, all pieces of these extraordinary systems have to be improved. We propose in this thesis different techniques to improve RJMSs by taking into account both administrator and user point of views. This novel approach leads to better performances and can lower the energy consumption.

Content and contributions

Chapter 2 presents works related to RJMSs. After a comparison of existing RJMSs, the most common RJMS scheduling heuristic for HPC (backfilling) is detailed. Then, major challenges bring to the forefront by the exascale goal are discussed. The previous points lead us to describe the various objectives considered in scheduling in HPC.

Once the RJMS field well understood, we present, in Chapter 3, several tools developed to perform experiments. The tools that were created and used to experiment our new RJMS scheduling strategies are introduced. We first developed

a methodology to replay existing traces of big cluster on a smaller cluster. This methodology is implemented in the *riplay* tool and is then used successfully in various experiments. We then go even further by creating a tool that simulates a cluster running one of the most used RJMS, SLURM. Lastly, we describe how to measure the energy consumption of a real cluster. We compare different techniques and show their accuracy. Experiments also show that depending on the software, the performance-energy trade-off is not obvious to determine.

Chapter 4 introduces a new scheduling strategy that provides the capability to autonomously adapt the executed workload to a limited power budget. The originality of this approach relies on a combination of speed scaling and node shutdown techniques for power reductions. It is implemented into the widely used resource and job management system SLURM. Finally, it is validated through large scale emulations using real production workload traces of the petaflopic supercomputer *Curie*.

In Chapter 5, we propose a job scheduling mechanism that extends the idea of the previous chapter. Again, it extends the backfilling algorithm, but this time, it is to become energy-aware while adapting resource management with node shutdown technique to minimize energy consumption whenever needed. This combination enables an efficient energy consumption budget control of a cluster during a period of time. The technique is validated and compared with various alternatives through extensive simulations. Experimental results show high system utilization and low bounded slowdown along with interesting outcomes in energy efficiency while guaranteeing the respect of an energy budget during a particular time period. Further works will improve this promising approach by coupling it with smarted algorithm techniques.

We developed algorithms to limit the energy or power consumption. However, a user has currently no incentive to employ them, as they might result in worse performance. In Chapter 6, we propose to manage the energy budget of a supercomputer through EnergyFairShare (EFS), a FairShare-like scheduling algorithm. FairShare is a classic scheduling rule that prioritizes jobs belonging to users who were assigned a small amount of CPU-second in the past. Similarly, EFS keeps track of users' consumption of Watt-seconds and prioritizes those whom jobs consumed less energy. Therefore, EFS incentives users to optimize their code for energy efficiency. Having higher priority, jobs have smaller queuing times and, thus, smaller turn-around time. To validate this principle, we implemented EFS in a scheduling simulator and processed workloads from various HPC centers. The results show that, by reducing its energy consumption, a user will reduce its stretch (slowdown), compared to increasing its energy consumption. To validate the general feasibility of our approach, we also implemented EFS as an extension for SLURM. We validated our plugin both by emulating a large scale platform and by experiments upon a real cluster with monitored energy consumption. We observed smaller waiting times for energy efficient users, which validates this new approach.

For now, we design several ways to improve performances using administrators knowledges or users knowledge. In Chapter 7, we investigate if both knowledges can be used at the same time thanks to a machine learning algorithm. While HPC systems generate an ever increasing amount of data, they are characterized by uncertainties on some parameters like the job running times. The question raised in Chapter 7 is: *To what extent is it possible/useful to take into account predictions on the job running times for improving the global scheduling?* We present in this chapter a comprehensive study for answering this question assuming the popular EASY backfilling policy. More precisely, we rely on some classical methods in machine learning and propose new cost functions well-adapted to the problem. Then, we assess our proposed solutions through intensive simulations using several production logs. Finally, we propose a new scheduling algorithm that outperforms the popular EASY backfilling algorithm by 28% considering the average bounded slowdown objective.

Conclusions and future work perspectives are finally presented in Chapter 8.

The studies presented in this dissertation have resulted in the following communications:

- Peer reviewed international conferences: [Geo+c], [Geo+b], [Geo+a], [Gau+]
- International communications:
  - "Road to exascale: what if end-users can help? An approach to respond to new system needs in the batch scheduler", *in* HPC Days in Lyon 2016, April 2016
  - "Improving Backfilling by using Machine Learning to Predict Running Times in SLURM", *in* Slurm Birds of a Feather (SC15), November 2015
  - "Power Adaptive Scheduling" and "Improving Job Scheduling by using Machine Learning", *in* Slurm User Group Meeting, September 2015
  - "Adaptive Resource and Job Management for limited power consumption" and "Introducing Energy based fair-share scheduling", *in* Slurm User Group Meeting, September 2014
- French communication:
  - "Ordonnancement dynamique des applications dans les supercalculateurs pour limiter la consommation électrique", *in* Green Days @Rennes, July 2014

Background

2

> *A picture is worth a thousand words. An interface is worth a thousand pictures.*
>
> — **Ben Shneiderman**

Before presenting contributions, we present in this chapter related works to this thesis. Each chapter contains a section for its own specific related works.

2.1 Resource and Job Management Systems

Resource and job management systems (also called workload managers, or resource and task managers, or batch schedulers) are middlewares that are responsible for delivering computing power to jobs. We already stated that they incorporate a scheduling algorithm to choose where and when jobs will run. Such software also implements a lot of other features related to the job management and resource management. These features include health checks, monitoring, profiling, accounting and managing library licenses.

There exists a large a variety of such software in HPC. Let us detail some of them.

SLURM [Yoo+03] is an open-source RJMS maintained by the company SchedMD. As of the June 2015 Top 500 computer list[1], SLURM was running on 6 of the 10 most powerful supercomputer.

OAR [Cap+05] is another open-source RJMS mainly developed by scientists at the Université Grenoble Alpes. Its main advantage is its high modularity: each component is totally independent and communicates through a database.

Flux [Ahn+14] is developed by the Lawrence Livermore National Laboratory and aims at supporting exascale clusters. This open-source RJMS uses a distributed key-value store to take distributed decisions.

Other noteworthy RJMSs are Condor [Lit+88], LSF [FD], PBS pro [Nit+04] and TORQUE [Sta06]. A complete list and history of RJMSs can be found in [Geo10].

2.2 Scheduling algorithms in HPC

Scheduling has been studied for centuries as it can improve how *things* are produced [Wea06]. Manufacturing is the first field that comes to mind, but even for computers this topic raised interests for decades [Gra66]. Unfortunately, most of these works are unusable because the assumptions are different in the field of HPC. For example, most theoretical works make the assumption that running times are

[1] `http://www.top500.org/list/2014/06/`

known in advance. In HPC, we only have an upper bound for this value. Another common issue is the scalability of such algorithms.

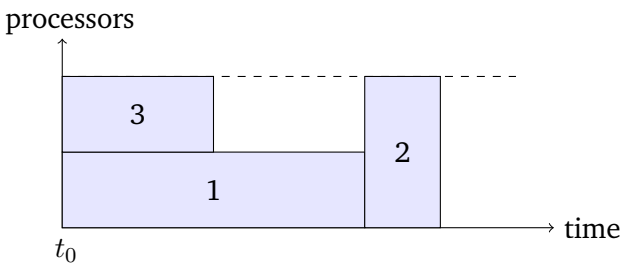


Figure 2.1. Example of backfilling for a queue of 3 jobs ordered according to their index.

Algorithm 1: EASY Backfill algorithm

```
for job ∈ queue do
    if job fits in system then
        launch job;
        remove job from queue;
    else
        exit loop;
    end
end
firstJob = pop first element of queue;
Make a reservation in the future for firstJob;
for job ∈ queue do
    if job fits in system then
        launch job;
        remove job from queue;
    end
end
Remove reservation for firstJob;
Push back firstJob at the top of queue;
```

These factors have lead engineers use greedy algorithms to resolve scheduling problems in HPC. Most of them are based on backfilling: the job priority is determined according to the arrival times of the jobs. Jobs are then scheduled in order. Then, the backfilling mechanism allows a job to run before another job with a highest priority only if it does not delay it. For instance, as it is depicted in Figure 2.1, where job's priority is their index, job 3 run before job 2.

Most available open-source and commercial job management systems use a heuristic approach inspired by backfilling algorithms. There exist several variants of this algorithm, like conservative backfilling [MF01a] and EASY backfilling [Lif95]. 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.

EASY is considered as an on-line algorithm since the decision to launch a job is only taken at the last moment. It is described on algorithm 1. It 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. This EASY backfilling policy is quite aggressive since only the first job in the queue cannot be delayed by backfilled jobs, which leads to an increased resource utilization rate. The popularity of this algorithm can then be explained by:

1. the ease of implementation,
2. the modularity,
3. the high resource utilization rate implied by this aggressive backfilling policy,
4. the scalability of being present-focused, and
5. the robustness regarding utilization rate.

As this kind of heuristic is very modular, RJMSs support a wide range of features, including re-ordering of the job list, reservation, and partitioning.

2.3 Exascale computing

2.3.1 Why do we need exascale supercomputers?

Bigger computations. Scientists want to simulate systems as their whole. For example, to study climate changes they want to simulate earth as a whole system.

Higher resolution. A higher resolution does not only mean more accurate results, it also means that new study can be performed. If scientists successfully simulate HIV at atoms level, it will open the possibility to look for new drugs to fight against it.

More data. Exascale will let us study bigger systems and analyze bigger problems. To perform basic gene studies terabytes of data need to be compared, exascale will open the way to new kind of statistical analysis.

2.3.2 The road to exascale

To reach an exaflopic system three main challenges need to be tackled [Don+11]. These challenges need to be addressed at both hardware and software level.

First, we need to develop algorithms and software that can exploit the massive parallelism of such systems. Most of the current developers have already parallelized a code, trying to minimize the overhead induced by the parallelization. Here a new order of magnitude in the parallelization has to be done. The goal is not to run on 8 or 32 cores but on billions.

Then, as already mentioned, exascale systems will have billions of components. Thus, the probability of failure of one of those is very high. The software and hardware have to be adapted to bypass these failures.

Last but not least, the energy consumption needs to be reduced. For the American Department of Energy, it needs to be reduced by a factor of 100 [Don+11]. On this thesis, we focus mainly on this hard challenge.

More precisely, we focus on how the RJMS may help in this challenge. At this level, only a few mechanisms are available to control and then reduce the energy consumption. Some resources may not be used, and even switched off, in order to reduce the energy consumption. But this comes at a high price: the resource cannot compute anything. Another mechanism is to use Dynamic Voltage and Frequency Scaling (DVFS). However, as shown in Section 3.4, the energy gain and the performances highly depends on the job run. This leads us to focus mainly on controlling the energy consumption.

2.4 Scheduling objectives

No perfect scheduling objective exists for RJMSs [FF05]. However, in this thesis, we will consider the following different measures.

The first measure is the utilization. The utilization is the percentage of processors that were used during a time period. This objective is mostly used by cluster owners as it depicts the productivity of the cluster.

Cluster owners may also want to measure the throughput. The throughput is the number of jobs that is started per second. This metric varies depending on the scheduling performances but also on the size of the jobs.

Instead of measuring the jobs, we can consider the user point of view. The waiting time is the time elapsed between its submission and its start. However, this metric does not take into account that user sees the time differently depending on the job length. A job that lasts 1 week can wait 1 day, while for a job that last 10 seconds this time seems enormous.

The stretch (or slowdown) improves the waiting time by taking into account the length of the job:

$$\text{stretch}_j = \frac{wait_j + p_j}{p_j}$$

where $wait_j$ is the waiting time of job j (from the time it is released in the system and the time it starts its execution) and p_j the running time of job j. The major drawback of this metric is the stretch of very small jobs. These jobs will have a very high stretch in realistic settings and, thus, the average (or the other norms) will be very high due to these small jobs.

A commonly admitted [Fei01] performance measure is the *bounded slowdown*, which is defined as follows:

$$\text{bsld}_j = \max\left(\frac{wait_j + p_j}{\max(p_j, \tau)}, 1\right)$$

where τ is a constant preventing small jobs to reach too large slowdown values. In the literature, τ is generally set to 10 seconds. We will use this reference value in the experiments.

Another related objective function usually used for comparing performances is the average of bsld, defined as:

$$\text{AVEbsld} = \frac{1}{n}\sum_{j} \max\Big(\frac{wait_j + p_j}{\max(p_j, \tau)}, 1\Big)$$

The third measure is of course the energy consumed. We have to be careful, sometimes the energy consumption is confused with power consumption. Let us remind the relation between them:

$$\int Power.dt = Energy$$

While instantaneous watt consumption is easy to measure, energy consumption is much harder to measure. Most of the time monitoring energy consumption is done by measuring the evoluton of the power in time. However, the frequency at which each measure is done is very important. Too many measures can disrupt the system while not enough measures will lack of precision. Challenges of energy measuring are discussed on the next chapter. Some mechanisms to overcome them are also presented.

Tools and methodology

3

> *Truth has nothing to do with the conclusion, and everything to do with the methodology.*
>
> — **Stefan Molyneux**

3.1 How to assess Resource and Job Management Systems?

3.1.1 Motivations

Technological evolution and scientific needs have made systems and applications more complex throughout the years and the study of the complete computing systems now depends on thousand of parameters and conditions. A large part of the research conducted in this field is mostly performed using simulator of the scheduling part of RJMS. These tools present advantages related to the control of experiments and the ease of reproduction but also limitations because they fail to capture all the dynamic, variety, and complexity of real life conditions [Bol+06].

The more advanced features are supported by the RJMS, the more complex gets the process of scheduling. Support of sophisticated algorithms such as backfilling and fair-sharing or features related to other abstraction layers like hierarchical resources, topology aware placement and energy consumption efficiency are all managed as extra parameters in scheduling and all induce an additional complexity upon the whole process. Furthermore, the analysis of the RJMS behavior is directly related to the conditions of usage such as workload and platform characteristics [FF05].

Hence, experimental methodologies are needed that will allow the experimentation of the actual RJMS for the study and evaluation of all its internal functions as one complete system, taking into account configuration parameters and usage conditions. Studying tradeoffs like energy efficiency versus system utilization, jobs turnaround times versus scheduling fragmentation, or system scalability versus jobs stretch times are some use cases that show how important is to study the system as a whole with no abstractions [Geo10].

The results obtained with our methodologies can be used as insights to compare the performances of different RJMSs, to improve the design and internals of a particular one, to tune the configuration of a production system in order to obtain better results for particular workload profiles, to experiment with new features upon the RJMS and finally to validate a real system before entering in production or after a maintenance period.

The complexity of the RJMS and the variety of the parameters and conditions that it can be used makes its behavior difficult to model and predict through a simulator of RJMSs. Specific simulators could only be useful to study a particular feature or observe a specific scenario, whereas we have a need for general methodologies that can be used for various use cases. This section and the next two introduces large-scale experimental methodologies that enable the study of the RJMS as one complete system.

3.1.2 Related Work

Performance evaluation is done by having the system scheduling a sequence of jobs. This sequence is the actual workload that will be injected to the system. Several works [Cha+99; CB01a; CB01b; FF05; FS07] have been carried out upon workload characterization and modeling of parallel computing systems.

In order to study a workload of a parallel system, Chapin et al. [Cha+99] have defined the *Standard Workload Format* (SWF) which provides a standardized format for describing an execution of a sequence of jobs. Some of the most important included fields are: job number, submit time, wait time, run time, number of allocated processors, status. A large archive of production HPC systems workload logs is maintained in the Parallel Workloads Archive [1].

There are two common ways to use a workload in order to evaluate a system.

1. Either use a *workload log*, also known as workload trace which is a record of resource usage data about a stream of parallel and sequential jobs that were submitted and run on a real parallel system [DF99],
2. or use a *workload model*, also known as synthetic workload, which is based on some probability distributions to generate workload data, with the goal of clarifying their underlying principles [CB01b; Won+00].

Based on previous works [Fei03; FF05], we can conclude that there is no definitive methodology that can provide precise and complete results. When a methodology (including the objectives) is selected, one should explain the aims, strength and weaknesses of it.

3.1.3 Content

In the next two sub-sections, two different methodologies will be presented. First, we developed a tool that replays workloads using a real RJMS on an emulated cluster. Then, we simulate the underlying layers of a RJMS (the system libraries) in order to run a unmodified RJMS upon a simulated environment. The last sub-section focuses on another important point to be able to perform our experiments: measuring energy consumption. We describe, in this sub-section, how to measure energy at large scale within the resource and job management system.

[1] `http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`

3.2 Large scale workload replaying

This section introduces a large-scale experimental methodology for Resource and Job Management Systems (RJMS, or batch scheduler) based on the replay of synthetic traces or real workloads extracted from production systems. This work has been done in collaboration with Joseph Emeras, Yiannis Georgiou and Olivier Richard. An important aspect of the methodology is the replay of the workload in the exact context that it was running in the original system. The methodology allows us to study particular aspects around scheduling without hiding the complexities of the rest of the system.

The next sub-section describes the steps of the methodology. Sub-section 3.2.2 discusses the limitations and sub-section 3.2.3 presents an implementation. Finally, sub-section 3.2.4 concludes this section.

3.2.1 Methodology steps

Selecting workloads

The evaluation of the RJMS can be driven with workloads derived from real platform logs or synthetic models.

Concerning the synthetic workloads, they are built from a statistical analysis of real and long workloads. ESP and Light-ESP [GH12] are examples of methodologies to built such synthetic workloads. As they are synthetic some details may be lost but the experiment is shorter and adaptable to different scenarios.

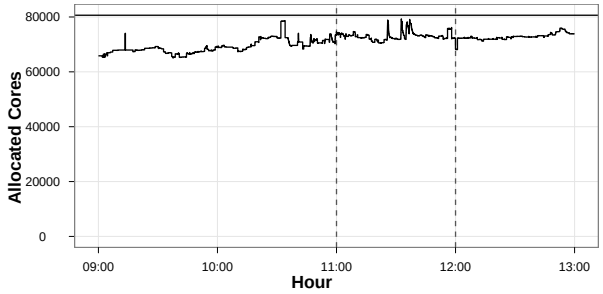
Another approach is to replay extracts of the original workload. Either we select representative extracts with statistical analysis or if the goal is to stress scheduling we have to choose interesting extracts. To select interesting extracts, from a large workload trace, several criteria have to be taken into account. First, the period of the extract should show a high utilization in terms of allocated resources and a high number of requested resources in waiting state. This ensures that the extract will belong to a period with high activity because issues such as scheduling fragmentation, job starvation, long stretch times, *etc.* surface mainly under stress. Moreover, there should be a high amount of jobs in order to allow the RJMS to make scheduling decisions.

The duration of the extract should be sufficiently short to not waste too many resources. It's possible to reduce the time of an extracted trace by a given factor but this will increase the overheads within the system.

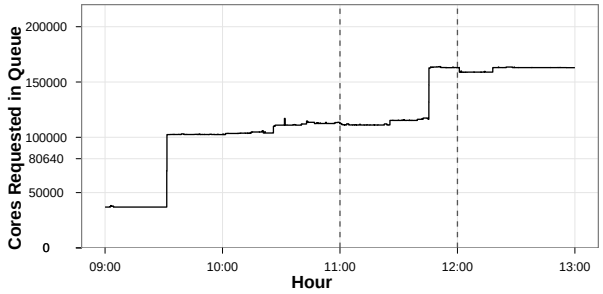
Platform emulation

Depending on the type of experiments, we may need to emulate the studied platform.

Emulation enables us to run the experiment without the need of an infrastructure similar to the original systems' size. Some RJMS provide emulation capabilities by allowing the usage of one physical node for multiple computing nodes. This is



(a) Utilization



(b) Queue

Figure 3.1. System utilization and Scheduling queue evolution on a workload trace (Curie,see footnote 1) around the selected trace. The dotted vertical lines delimit the selected trace. No comparison can be made with replayed traces, see paragraph 3.2.2.

done by allowing the usage of different ports upon one IP address to represent each emulated node.

Since our focus is the study and evaluation of the RJMS software, applications can be also emulated. As long as the job does not interact with the RJMS, a simple *sleep* command is enough.

Platform and job emulation allow us to evaluate large-scale systems using a small infrastructure. This enables the methodology to scale up to platforms' sizes that may even not exist yet.

Environment Initialization

At this step, the RJMS has to be set in a state as close as possible as it was in the original run. Here are the most important parameters that have to be taken into account to initialize a replay of a workload:

- Inject Queued and Running Jobs. Imitating the activity of the original run, queued and running jobs have to be injected within the RJMS.
- Configure Placement of Running Jobs. The resources selection for running jobs is based upon the original placement in order to follow the same fragmentation.
- Initialize Internal States. RJMS store internally different counters and cached computation. For example, most fairshare implementations depend on stored values.

Workload Replay

Once all preparations are done the testbed is ready for experimentation and the selected workload can be replayed. Jobs should be submitted exactly as they were from the original trace, respecting workload characteristics such as running times, requested resources, reservations, jobs ownerships and other. The duration of the workload replay is equal to the duration of the original extracted workload. In the case of synthetic workloads, this step ends when the last job is finished.

3.2.2 Limitations

There are limitations in the implementation of the methodology. All events that may occur on the platform should be taken into account which is nearly impossible.

For example, machine failures may impact considerably the platform and in addition, this information is not always available. Possible improvements can include a machine failure model [Kon+10; SG06].

If the platform is emulated, the limitations of the emulated environment have to be taken into account, like IO bandwidth limitations, overloaded CPU etc.

Furthermore, user behavior is impacted by the system response time, as studied by Shmueli et al. [SF07]. The aim of this methodology is not to reproduce an actual cluster run, but to compare and predict system performance on different setups.

3.2.3 Implementation

We have implemented our methodology in a dedicated tool called *riplay*. On the technical aspects, the tool takes as input a trace in SWF format, extracts the useful information and submits the sequence of jobs from the trace to the RJMS. The order of arrival, the inter-arrival times, the resources and the time requests of the jobs are respected during the replay.

Replayed jobs are just composed of *sleep* commands that wait for the original job duration. *riplay* takes into account the environment setup. When a trace replay is launched, the first step is the environment reconstruction. Internal values of fairsharing are computed from the original workload and then injected into the RJMS. Once this is done, the tool submits all the jobs that were running. In order to ensure that they will end their execution at the same time they had in the original workload, their time characteristics are modified. Then jobs that were queued are submitted. Finally, the execution of the workload starts.

3.2.4 Conclusion

This tool has been used in several papers [Geo+c; Geo+b; Geo+a] and is used internally to test RJMSs and assess hypothesis.

The main drawback of this approach is the time taken to simulate a trace. As one of our aims is to test our algorithms on huge machines, we cannot shrink time too much otherwise it will induce network contention when a lot of jobs finish or start at the same time. This motivates us to go further than just emulate the platform. In the next section, we present our platform simulator.

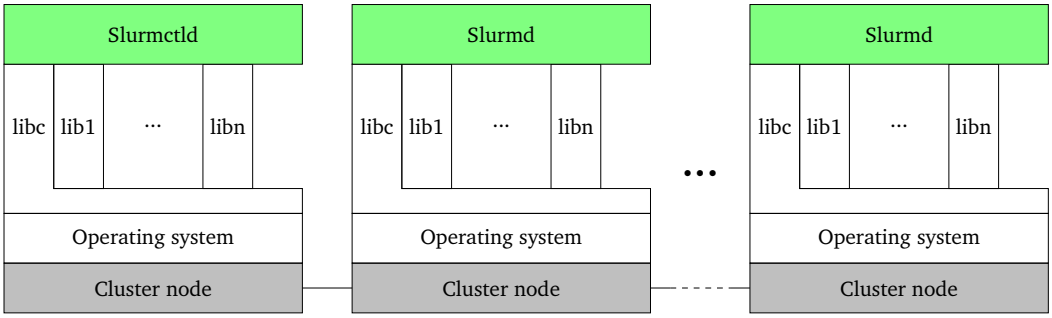


Figure 3.2. SLURM software and hardware stack.

3.3 Simunix, a platform simulator

Because RJMSs are distributed systems designed to run with a large amount of cores and nodes, testing them is not an easy task. A simulator will make testing them easier. However, as stated in Section 3.1.2, simulators are limited. Thus, we take the decision to not develop a simulator of the RJMS itself, but to simulate the cluster in which it runs. This work has been done in collaboration with Adrien Faure.

We created a platform simulator with the following features:

- no modification to the RJMS's code,
- simulate most actual HPC cluster,
- scale up to exaflopic cluster,
- reproducible results and
- the possibility to speed up time.

Such simulator is a great tool for many use cases. Firstly, it would provide an easy way to test RJMSs without deploying it on a real HPC cluster. Secondly, RJMS's developers may use it to test new features such as new scheduling algorithms. Thirdly, RJMSs offer many parameters to fit with user's requirements. This simulator would give a quick and easy way to find a suitable configuration for HPC clusters' administrators. Finally, the capability to speed up time will allow studying traces of long periods of times in order to evaluate the scheduling performances.

To assess our approach, we implement a tool called *Simunix* that works with RJMSs that use a standard *libc* on a 64bit Linux operating system. We first focus on running SLURM. More RJMSs will be usable later, but it is easier to focus first on one RJMS.

3.3.1 Architecture

Simunix is designed to fit all requirements introduced in the previous section. First, we use a simulation framework to simulate the underlying HPC cluster. Next, we isolate all simulated processes into a wrapper. Then, the wrapper intercepts the functions we need to simulate, called by the simulated process. The functions we need to simulate are the functions related to time (*e.g.* `sleep()`, `gettimeofday()`) and the functions that could block the execution of the process

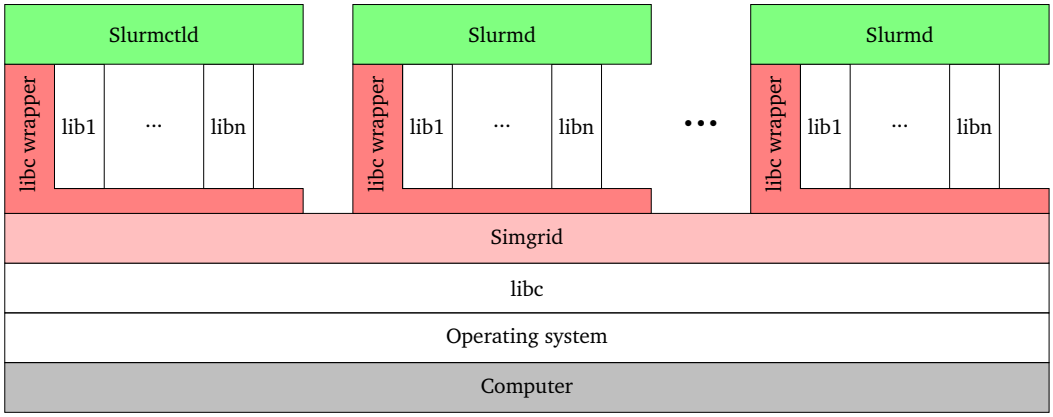


Figure 3.3. SLURM software and hardware stack when using Simunix.

(*e.g.* `pthread_join()`, `recv()`). Hence, wrappers synchronize their executions with the simulation framework.

This framework is Simgrid [Cas+14], a framework to design simulators of distributed systems. Simgrid supports advanced networks models, an energy consumption model and an I/O model. Moreover, Simgrid is actively developed and well tested. They validate and invalidate their simulator, they explicitly give the strengths and weaknesses of their models by testing them and compared them to real runs. It makes Simgrid a perfect candidate for our simulation engine.

Wrappers are separated processes that intercept time or blocking functions and translate them to Simgrid calls. To intercept function calls, wrappers modify the Global Offset Table (GOT) of each program before their executions. The GOT is the table of the addresses of global objects (like global variables or functions) which resides in the data section of an ELF (Executable and Linkable Format) file. External (or *shared*) functions are global objects and thus lies in the GOT. The wrapper modifies entries of interesting functions at the start time of each simulated executable.

Finally, Simgrid implements a function that can simulate arbitrary amounts of computations and network transfers. Jobs are replaced by this function. This method creates opportunities to shrink time while simulating cluster usage.

3.3.2 Limitations

First, this project is not finished yet and thus a number of questions related to its performance are still open. Its performance is measured by the speed of simulations. Interception, translation, and simulation are crucial steps to efficiently simulate clusters. How fast we can perform these operations and the resources needed for them will be one of the focus of the evaluations of our simulator.

Then, simulator results have to be as close as possible to the results of real runs. This task depends on the simulator itself but also on the calibration of it. We have to give instructions on how to make these calibrations and how to determine if the simulations are close to real runs.

The last limitations are related to the interception technique. A program can choose to perform system calls directly to the kernel without going through the libc. At the moment, we do not intercept system calls and these calls may block the simulation. Also, resources related calls are not intercepted either (like getting the energy consumption using the library FreeIPMI). If they are done using a library, it can be added to our simulator. But, if this is done by reading a register or a specific memory location, we cannot intercept them.

3.3.3 Conclusion

This ongoing work will improve our scientific and engineering workflow. It will enable us to test quickly SLURM on machines with scales, heterogeneity, and complexity that don't even exist. Before using it, a number of tests will be necessary to be sure that our simulator has a behavior close enough to the reality.

3.4 Energy Accounting and Control with SLURM Resource and Job Management System

One of the main focus of this thesis is the energy consumption of HPC platforms. As far as the systems middleware concerns, the Resource and Job Management System (RJMS) can play an important role since it has both knowledge of the hardware components along with information on the users workloads and the executed applications. Energy consumption is the result of application computation hence it should be treated as a job characteristic. This would enable administrators to profile workloads, users to be more energy aware, and researchers to better understand energy consumption is such systems. Furthermore power consumption analysis with timestamps would enable users to profile their applications and perform optimizations in their code. Hence the first step that needs to be made is to measure power and energy through the RJMS and map them to jobs. Introducing Power-Meters on Supercomputers would be too expensive in terms of money and additional energy hence using the already available interfaces seemed the most viable approach.

This section presents a paper [Geo+c] which describes the design and evaluation of energy accounting and control mechanisms implemented upon the SLURM Resource and Job Management System. This work has been done in collaboration with Yiannis Georgiou, Thomas Cadeau, Danny Auble, Morris Jette, and Matthieu Hautreux. This work is used in various experiments along this thesis.

SLURM has a particular plugin dedicated to gathering information about the usage of various resources per node during job executions. This plugin, which is called *jobacct_gather*, collects information such as CPU and Memory utilization of all tasks on each node and then aggregates them internally across all nodes and returns single maximum and average values per job basis.

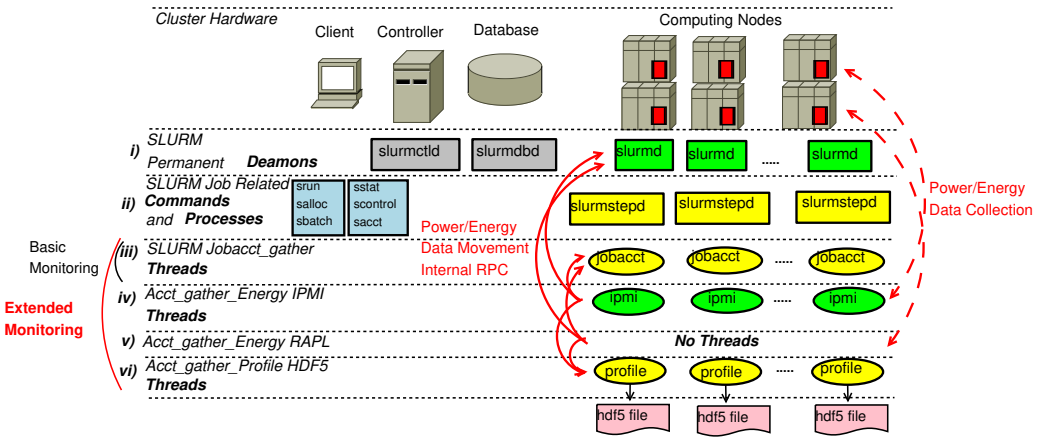


Figure 3.4. SLURM Monitoring Framework Architecture with the different monitoring modes

Resource utilization collection was already supported in SLURM. The goal of our new developments was to extend this mechanism to gather power and energy consumption as new characteristics per job. Of course, as we will see in this section, only per job energy consumption is stored in the SLURM database. Instant power consumption needs to be stored in relation with timestamps and thus a different model is used.

3.4.1 SLURM Monitoring Framework Architecture

We will begin by delving a bit more into the details of SLURM's architecture of the monitoring mechanisms. Figure 3.4 shows an overview of SLURM's daemons, processes and child threads as they are deployed upon the different components of the cluster. In the figure, the real names of daemons, processes and plugins have been used (as named in the code) in order to make the direct mapping with the code. In SLURM architecture, a job may be composed by multiple parallel sub-jobs which are called steps. A job is typically submitted through the `sbatch` command with the form of a bash script. This script may contain multiple invocations of SLURM's `srun` command to launch applications (sub-jobs) in parts or the entire job allocation as specified by `sbatch`. To simplify the explanation let us consider the simpler case of one job with one step. When this job is launched, the `srun` command will be invoked on the first node of job allocation and the slurmd daemons will launch a slurmstepd process on each node which will be following the execution of the whole step (shown by latin number ii) in figure 3.4).

If the basic monitoring mode is activated the *jobacct_gather* plugin is invoked and the slurmstepd process will launch a thread upon each node that will monitor various resources (CPU, Memory, *etc.*) during the execution (as shown by iii) in Figure 3.4). The polling of resources utilization is done through Linux pseudo-file system /proc/ which is a kernel data structure interface providing statistics upon various resources in the node. The sampling frequency is user specified on job submission. The various statistics are kept in data structures during the execution

and aggregated values upon all nodes (Average,Max, *etc.*) are stored in the database when the job is finished. The data are moved between threads and processes with internal RPC keeping the mechanism reliable and efficient. This information can be retrieved by the user during the execution of the job through the SLURM command `sstat` and after its termination through `sacct`.

The new monitoring modes will follow the same design architecture with separate plugins for each monitoring mode and launching of new threads to keep their polling asynchronous with the basic monitoring mode.

3.4.2 Power data collection through IPMI interface

The Intelligent Platform Management Interface [Int] is a message-based, hardware-level interface specification conceived to operate even independently of the operating system in an out-of-band function. It is used by system administrators mainly to perform recovery procedures or monitor platform status (such as temperatures, voltages, fans, power consumption, *etc.*) The main intelligence of IPMI is hidden on the baseboard management controller (BMC) which is a specialized microcontroller embedded on the motherboard of a computer which collects the data from various sensors such as the power sensors. The advantage of IPMI is that it can be found in nearly all current Intel architectures and if power sensors are supported it can provide a very cheap built-in way for power collection. Various open-source software[2][3] exist for in-band and out-of-band collection of IPMI sensors data.

The energy data are communicated between the IPMI thread towards jobacct thread and slurmd using internal Remote Procedure Calls. The instant power data are communicated with another RPC towards the profile thread if profiling is activated, as show us the red arrows in Fig.3.4.

3.4.3 Energy data collection from RAPL model

The Running Average Power Limit (RAPL) interface has been introduced with the Intel Sandy Bridge processors [Rot+12] and exists on all later Intel models. It provides an operating system access to energy consumption information based on a software model driven by hardware counters. One of the specificities of the model is that it tracks the energy consumption of sockets and DRAM but not that of the actual memory of the machine.

Based on the fact that the actual call to RAPL is fast and the fact that we do not need separate sampling there is no need to have a particular thread responsible for collecting the energy data on each node (as shown in Fig.3.4 with latin number v)). Hence the dedicated functions of the *acct_gather_energy* RAPL plugin collect data from the RAPL model whenever they are asked and transfer those data through internal RPC (red arrows in Fig.3.4) towards the demanding threads or processes.

[2] http://ipmitool.sourceforge.net/
[3] `http://www.gnu.org/software/freeipmi/`

3.4.4 Power data Profiling with hdf5 file format

Energy consumption is a global value for a job so it is obvious that it can be stored in the database as a new job characteristic. However, power consumption is instantaneous and since we are interested in storing power consumption for application profiling purposes; a highly scalable model should be used. Furthermore, since the application profiling will also contain profiling statistics collected from various resources (like network, filesystem, *etc.*), a mechanism with extensible format would be ideal to cover our needs.

Hence a new plugin has been created called *acct_gather_profile*, presented by latin number vi) in Fig.3.4. This plugin is responsible for the profiling of various resources. Apart energy profiling, network, filesystem, and tasks profiling are supported but the analysis of these types goes beyond the purposes of this chapter. As we can see in the graphic representation of Fig.3.4 the profile thread is launched from slurmstepd (same color) and it collects the power and energy consumption data from IPMI thread or RAPL functions through internal RPC. The collected data are logged on hdf5 [Fol+99] files per node. Ideally, the storage of these files should be made upon a shared filesystem in order to facilitate the merging of files after the end of the job. The profiling information cannot be retrieved by the user during the execution of the job and this is because the hdf5 files are binaries which are readable only after the job termination.

3.4.5 Evaluation

We evaluated the overhead of the framework by executing the Linpack benchmark and our study showed that the cost of using the framework is limited; less than 0.6% in energy consumption and less than 0.2% in execution time. In addition, we evaluated the precision of the collected measurements and the internal power-to-energy calculations and vice versa by comparing them to measures collected from integrated Wattmeters. Our experiments showed very good precision in the job's energy calculation with IPMI and even if we observed a precision degradation with short jobs, newer BMC hardware showed significant improvements. Hence our study shows that the framework may be safely used in large scale clusters such as Curie and no power-meters are needed to add energy consumption in job accounting.

The full evaluation of this approach can be found on [Geo+c].

3.4.6 Performance-Energy Trade-offs

In this section, we are interested in the observation of performance-energy trade-offs with particular applications. The tracking of performance and energy consumption can be made either during the execution of particular commands (sstat or scontrol show node) or at the end of the job with results directly stored in the SLURM database or on hdf5 files dedicated for profiling. Furthermore, this tracking is not passing from the overhead of the integration of a dedicated profiling program but rather through a direct integration of sensor polling inside the core of SLURM

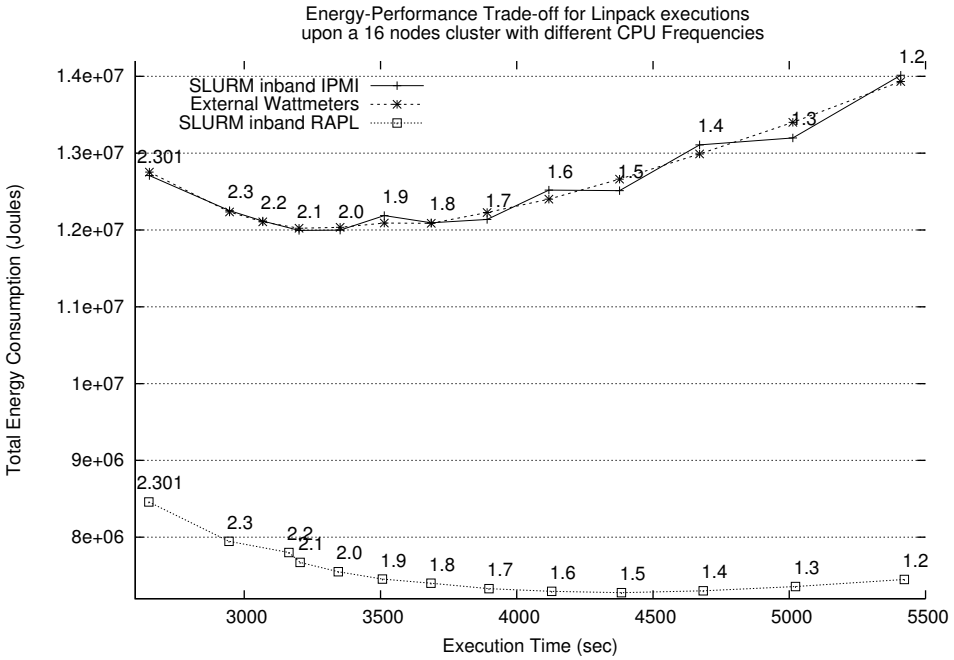


Figure 3.5. Energy-Time Tradeoffs for Linpack benchmark at different CPU frequencies

code. Hence this section shows the type of research that can be possible using the extensions on SLURM for monitoring and control of energy consumption and how these features can be helpful in order to make the right choice of CPU Frequency for the best performance-energy trade-off. We make use of three different benchmarks (Linpack, IMB and stream) and make various runs by changing the CPU Frequency and compare the overall energy consumption of the whole job.

The figures that follow show the performance-energy tradeoffs with different CPU Frequencies. Figure 3.5, 3.6 and 3.7 show us the runs for respectively Linpack, IMB and Stream benchmarks.

Linpack benchmark shows that the lowest energy consumption is reached using 2.1GHz and in fact, this is the best tradeoff between energy and performance. We can also observe that while the energy consumption is dropping until this frequency it kind of stabilizes then with higher executions times until 1.7 GHz but increases after that. It is interesting to see that dropping the frequency lower than 1.4 has no benefit at all because we lose both in energy and performance in comparison with the turbo mode. Between IPMI and Wattmeter graphs there are some small differences but in general, the IPMI follows closely the same tendencies of Wattmeters which proves that we can trust this kind of monitoring for this type of energy-performance profiling too. RAPL graph for the Linpack benchmark (left) shows us a kind of different behavior with best tradeoff performance-energy between the frequencies 1.6 and 1.5. In addition, we do not see the same tendency like the Wattmeter or IPMI graphs. This is related to the fact that RAPL only monitors a partial view of the active elements of the node, elements for which the energy consumption is mostly related to the frequency (and thus voltage) of the cores. The static consumption of

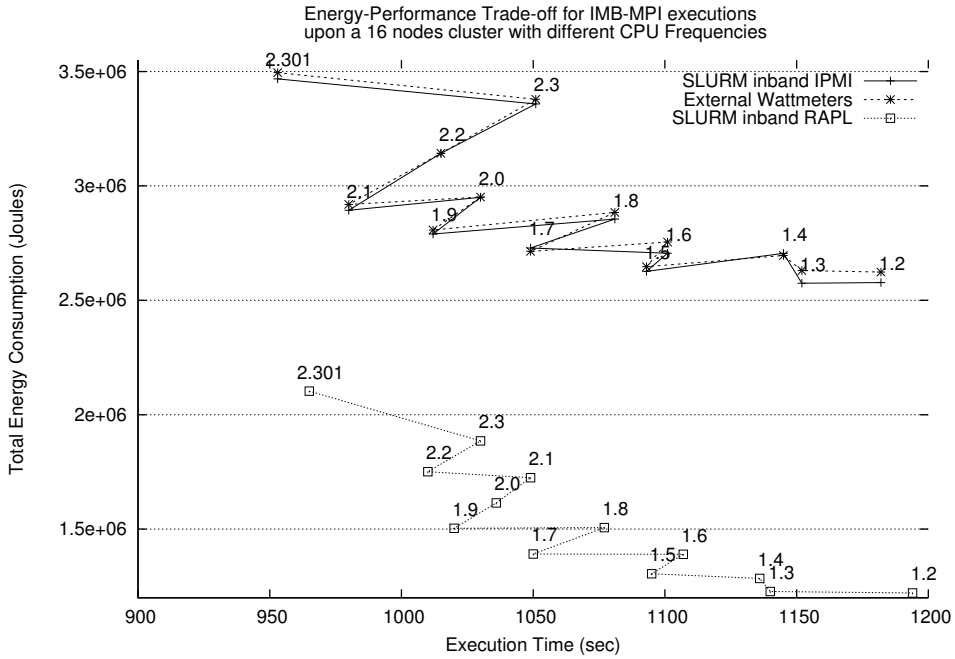


Figure 3.6. Energy-Time Tradeoffs for IMB benchmark at different CPU frequencies

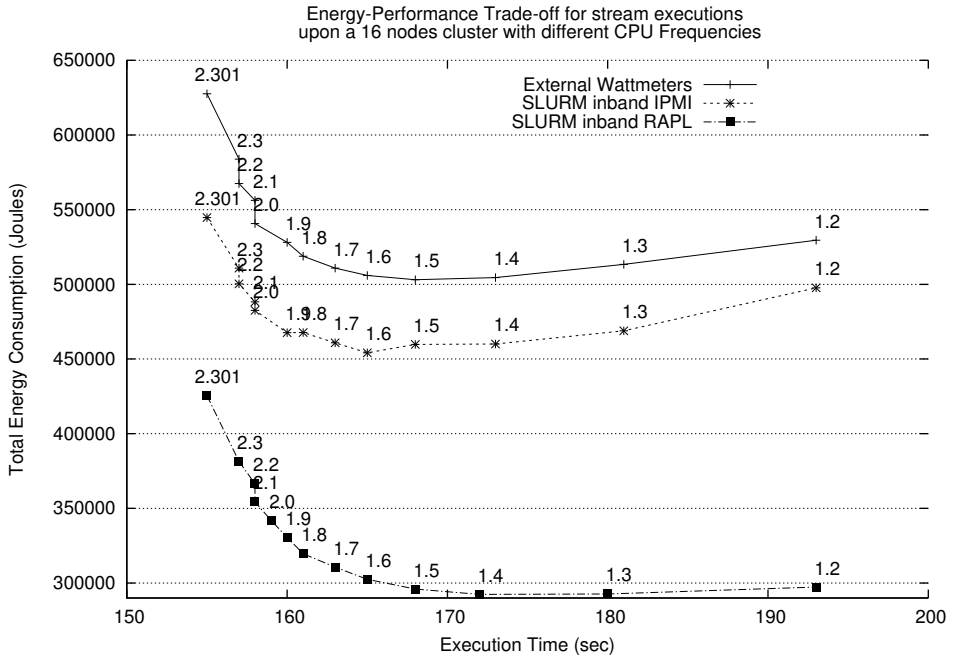


Figure 3.7. Energy Time tradeoffs for Stream benchmark at different CPU frequencies

the nodes, including the disks, network cards, motherboard chipsets and every other electrical components or fans are not taken into account in RAPL. Increasing the computation time of an application involves increasing the share of the energy usage of all these second tier elements up to counter-balancing the benefits of running at a lower frequency.

Finally on figure 3.7 we can see graphically the error deviation of IPMI vs Wattmeters. We can observe that the best tradeoff is performed with Frequencies 1.6 or 1.5 . It is interesting to see how this graph has an obvious logarithmic design with stability on execution times and changes on energy from 2.301GHz until 1.6 and stability on energy consumption and changes on execution times from 1.6 until 1.2.

Overall, it is interesting to see how each application has different tradeoffs. That is the reason why particular profiling is needed for every application to find exactly the best way to execute it in terms of both energy and performance. The outcome of these evaluations is to raise the importance of energy profiling to improve the behavior of the applications in terms of performance and energy and show how the new extensions in SLURM can be used to help the user for this task and find the best tradeoffs in energy and performance. Furthermore, the comparison of monitoring modes with Wattmeters proves that SLURM internal monitoring modes provide a reliable way to monitor the overall energy consumption and that controlling the CPU Frequency is made correctly with real changes on performance and energy. We argue that most real life applications have execution times far bigger than our long running jobs here (Linpack) so the energy performance tradeoffs will definitely be more important than reported in this section.

4 Adaptive Resource and Job Management for Limited Power Consumption

> *The realisation that limitations are imaginary will make you strong and overpowering.*
>
> **— Stephen Richards**

In this chapter, we propose a powercapping mechanism implemented in SLURM, based on a combination of both DVFS and node shutdown power-reduction techniques. This chapter corresponds to a publication [Geo+b] made in collaboration with Yiannis Georgiou and Denis Trystram.

We study a generic model which shows that in some cases, the best solution is to mix both techniques. As a consequence, the RJMS has to be adequately adapted in order to efficiently schedule the jobs with optimized performance while limiting power usage whenever needed. More precisely, the main contribution is the introduction of a new power consumption adaptive scheduling strategy that provides the capability to autonomously adapt the executed workload to the available or planned power budget. We have studied the impact of DVFS on several actual applications along with the effects of using grouped shut-down of nodes and considered them in our model. The new scheduling algorithms have been implemented in the widely used open-source workload manager SLURM. As our aim is to integrate this mechanism into large-scale supercomputers such as *Curie*, we have validated our model, algorithms, and implementations using large-scale experimentations based on real production workload traces of the *Curie* petaflopic supercomputer.

4.1 Related Works

4.1.1 Controling the power

Dynamic Voltage and Frequency Scaling (*DVFS* in short) is the most popular mechanism used so far for controlling the power in computing systems and as a consequence, reduce the energy. There exist a lot of works oriented toward theoretical results on speed-scaling (for instance continuous speeds). We are targeting here more realistic issues. The first series of papers intended to determine the right frequency. Energy-centric DVFS controlling method was proposed in [Kim+12] for single CPU multi-core platforms. The idea was to monitor the traffic of data from the cores to the memory and to update the DVFS accordingly (i.e. reduce it if the traffic is high or expand if it is low). This was extended in [Sno+05] for more

general platforms. Schöne and Hackenberg [SH11] used register measurements for determining the frequency. Kimura et al. [Kim+08] provided also a new mode of control of DVFS through a code-instrumented DVFS control. Then, Gandhi et al. [Gan+09] considered a mechanism that switch between the maximum DVFS to the idle state, in order to minimize the energy consumption. DVFS has also been studied to predict its impact on the whole system. Rountree et al. [Rou+11] developed a performance prediction model outperforming previous models. Etinski et al. [Eti+12b] studied how to improve the trade-off energy versus completion time on applications. Freeh et al. [Fre+07] provided a huge number of experiments for measuring the Energy over Time. An interesting feature was studied in the case where a node is removed from the system (moldable jobs).

Another complementary mechanism consists in switching off some nodes (also called *shutdown*). Lawson et al. [LS05] proposed an opportunistic shutdown mechanism, which switches off a node after a significant idle period. Aikema et al. [Aik+11] studied the energy from the view point of a cost function. Here, a node which becomes idle is considered as a zero-cost in term of energy. Under the assumption of under-loaded cluster, Hikita et al. [Hik+08] presented a batch scheduler that minimizes the number of active nodes while keeping the same performances. In [Dem+07], Demaine et al. took into account both the cost of energy and of switching (off/on) the processors. They proposed a theoretical algorithm that minimizes the number of such switches.

Despite the two first mechanisms, other options have been studied including network frequency scaling [Sha+03] or temperature-aware scheduling [Moo+05]. An incentive mechanism for reducing energy has also been proposed in [SV09].

4.1.2 Powercapping

In the following of this chapter, we are focusing on powercap as the main topic. Some papers are considering powercapping at the node level, for instance [Rou+12a] used a new feature available in Intel processors to achieve a local powercap and [Red+12] packed threads together in order to tune the DVFS. Powercap has been studied by Etinski et al. in a series of papers [Eti+10a; Eti+10b; Eti+12a] where DVFS is the only mechanism used to achieve powercapping while keeping good performances. We consider here a more sophisticated mechanism including also shutdown.

In the context of cloud computing, Geronimo et al. proposed a virtual machine manager that can use DVFS, update the virtual machine resources, migrate them and shutdown opportunistically some processors [Ger+14]. Fan et al. defined in [Fan+07] a methodology to reduce the global cost of data-centers by buying more processors and capping their power consumption. Our work differ mostly from the constraints: in this chapter we only take into consideration HPC supercomputers.

To the best of our knowledge, there is no similar work which consider DVFS and shutdown simultaneously in order to adapt the power consumption of a HPC cluster. Pierson and Casanova proposed a theoretical approach based on mixed Integer Linear Programs restricted to a single application [PC11]. We propose here a generic mechanism which selects the best strategy among DVFS, shutdown or both together.

4.2 Energy and Power analysis

In this section, we describe a new model that enables to determine when to switch off nodes and to determine the right frequency.

4.2.1 Tradeoff Switch-off between DVFS

Let us define W as the maximum amount of computations that could be performed during a given period of time T:

$$W = T \times (\frac{N - N_{off} - N_{dvfs}}{1} + \frac{N_{dvfs}}{deg_{min}}) \quad \text{(C1)}$$

Where N is the total number of nodes; N_{off} is the total number of nodes which are switched off and N_{dvfs} is the total number of nodes whose frequency has been decreased.

deg_{min} represents the percentage of *degradation* of the completion time at the minimum frequency compared to the maximum one. The justification to take deg_{min} at the minimum frequency is to consider the maximum possible impact on applications. This choice will be discussed in Section 4.5.

Obviously,

$$N_{dvfs} + N_{off} \leq N \quad \text{(C2)}$$

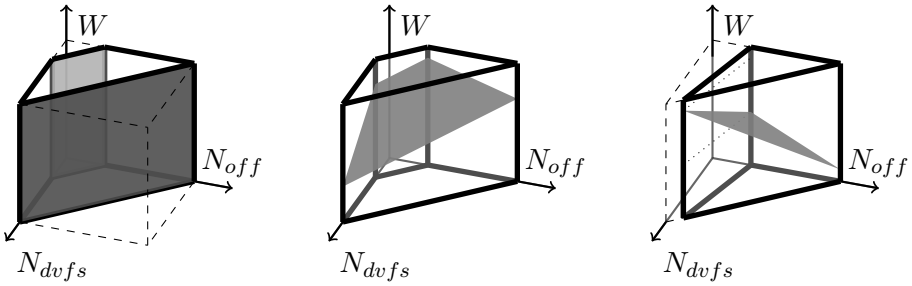
Without loss of generality, T will be set to 1.

The consumed power should be lower than the powercap:

$$\begin{aligned} &N_{off}.P_{off} + N_{dvfs} \times P_{min} + \\ &(N - N_{off} - N_{dvfs}) \times P_{max} \leq P \end{aligned} \quad \text{(C3)}$$

Where P_{off}, P_{min} and P_{max} are respectively the power consumed by a switched-off node, by a node in the lowest frequency and by a node in the maximum frequency. P is the powercap, *i.e.* the global available power at this moment.

The previous constraints C1 and C3 correspond respectively a plane and an half-space in the 3 dimensional space (W, N_{off} and N_{dvfs}). We are looking for points that maximize W within the previous constraints. The intersection of the two



(a) (C2) constraint is represented by the dark grey area while ligth grey represents (C3) constraint.

(b) (C1) equation under the constraints is represented by the grey area.

(c) (C1) equation under the constraints is represented by the grey area.

Figure 4.1. Visual representation of our model.

previous surfaces is a segment. We then consider the point that maximizes W on this segment. Then, there are four cases to distinguish:

1. There is only some switched-off nodes (the best point is on located on the plane (W, N_{off}).
2. There is only use of DVFS on some nodes (the best point is on located on the plane (W, N_{dvfs}).
3. Both previous options lead to the same maximum computational load (all the points of the segment are eligible)
4. The powercap is too low and both mechanisms should be used to reach the maximal W (the best point is at the intersection of the segment and the constraint C2).

The value for the two first cases can be easily computed thanks to the following formulas:

$$\begin{cases} N_{off} = \frac{P - N \times P_{max}}{P_{off} - P_{max}} \\ N_{dvfs} = 0 \end{cases} \quad \text{or} \quad \begin{cases} N_{off} = 0 \\ N_{dvfs} = \frac{P - N \times P_{max}}{P_{min} - P_{max}} \end{cases}$$

Let W_{dvfs} be available computational load available using only the DVFS mechanism (similarly W_{off} for the switch off mechanism). DVFS is better to use when $W_{dvfs} > W_{off}$. Which is equivalent to $\rho > 0$, where $\rho = 1 - \frac{1}{deg_{min}} - \frac{P_{max} - P_{dvfs}}{P_{max} - P_{off}}$. In the third case, both mechanisms give the same results. Thus, for the three first cases, it is easy to determine which mechanism to use depending on ρ. Let us focus on the last case when the powercap is too low. How low this powercap has to be to reach this limit? The powercap is too low when $P < N \times P_{off}$ or $P < N \times P_{min}$. The first expression can not happen practically since the powercap will be less than the clusters completely switched-off. Let us define $P = \lambda N P_{max}$, where λ is the powercap normalized by the maximum power consumption of the cluster. The second expression becomes $\lambda < \frac{P_{min}}{P_{max}}$, which means that the powercap can not be less than $\frac{P_{min}}{P_{max}}$ if DVFS is the only mechanism used to control power. In this case, the best choice for N_{off} and N_{dvfs} is:

$$\begin{cases} N_{off} = N - N_{dvfs} \\ N_{dvfs} = \frac{P - N \times P_{off}}{P_{min} - P_{off}} \end{cases}$$

We are now able determine which mechanism to use depending on the job, the cluster and the powercap. In the following section we will present an optimization of the switch-off mechanism. Then we will discuss the algorithm from the the described model.

4.2.2 Power Bonus when switching off hardware components

In HPC clusters, there are several hardware levels from a power consumption point of view. A level is defined as a group of different hardware components that can be switched-off simultaneously.

The configuration of which hardware components participate at each level depends on the architecture of the cluster. For instance, if the architecture is such that nodes are grouped in order to mutualize the first layer of administration and interconnection networks switches, nodes belonging to a same group can be powered off along with their respective switches without preventing the correct usage of the remaining groups. In this example, such a group will compose a particular power level. The extra power consumption gained by the network switches when powered off is then called a 'power bonus'. This method allows us to reduce even more the power consumed by a cluster when disabling part of its compute power.

In modern architectures, typical HPC clusters will have different 'power bonus' related to the different levels of components aggregation. Correctly selecting the computing elements to switch-off when coping with a power constraint will thus enable to sum up the different bonus at the different levels and maximize the power available to the active compute elements and their hardware dependencies. The lowest level considered in the our model is the multicore node. No actual power bonus is currently provisioned at this level. Individual cores and sockets can not be switched-off individually, hence they cannot comprise a lower level on their own.

In Section 4.5.1 we provide more precise details for the 'power bonus' related to the Curie cluster architecture.

4.3 Scheduling under power control

4.3.1 Preliminaries on scheduling features

From a high level point of view, scheduling in a Resource and Job Management System can be decomposed into two successive phases: first, the jobs should be selected after prioritization among the group of pending jobs, then, the resources should be selected upon which the job will be dispatched. In more detail, the first phase may involve various mechanisms to select the next job to be treated. For instance, the usual backfilling [MF01b] may be enriched with multifactor priorities such as job age and job size or even more sophisticated features like fair-sharing

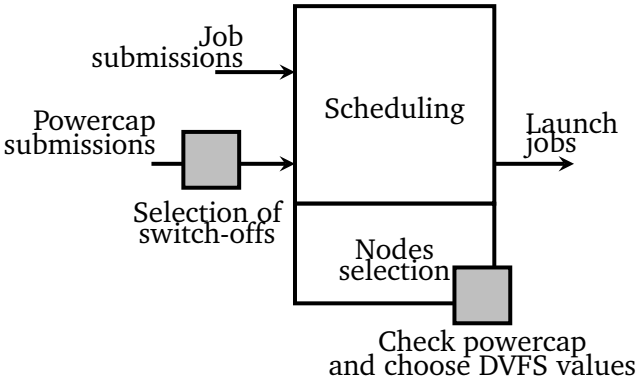


Figure 4.2. SLURM architecture. In grey, modified part.

and preemption. The second phase is related to the actual allocation of resources according to their characteristics such as internal node topology, network topology, usage of accelerators. The proposed powercapping algorithm takes place during this second phase of scheduling. One of the main building blocks of this algorithm relies on the fact that power is treated as a new type of resources characteristics. According to its state (PowerDown, Idle, Busy, etc.), the resource will consume a different amount of power. At any moment, the RJMS keeping the state of each resource internally can deduce the power consumption of the whole cluster. The characteristic of power can be related to any different component of the cluster but for the sake of clarity, we consider here the power of a whole node.

The calculation of the power consumption of the cluster is simply obtained by summing up the power consumptions of each node. For instance, the nodes that are executing jobs will be counted with the maximum power consumption (except in cases of DVFS usage); the nodes that are kept idle will be counted with the minimum power consumption and those that have been switched-off are counted with no power consumption (it could be non-zero in case where the BMC (Baseboard Management Controller) is still on). The algorithm goes one step further by considering the setting of the different CPU frequencies as different power states. Hence, the power consumption of each node will change depending on the CPU frequency at which the job is running.

The power values related to the state of each node can either be measured or be given by the constructor (this information can be configured and kept internally into the RJMS).

4.3.2 Scheduling algorithm for powercapping

The powercapping algorithm that we propose is composed of two successive parts: A first part where the decisions for power management are taken in advance (to better prepare future actions) followed by a part where the power distribution and management take place.

The algorithm is activated as soon as a powercap reservation is provided. This powercap value can be either set for *now* (i.e. the moment when the command

is launched) with no time restriction/limitation or as a reservation for a certain time window in the future. When a new job arrives during the allocation phase, the algorithm examines if there is any powercap limit for the time being or if the job may overlap with any future reservation of power at some point in the future. If any of these cases holds, the power consumption of the cluster is computed by considering as busy the nodes that the job will use. Then, the different values of the a-posteriori power consumption of the cluster are examined by measuring the power with all the different CPU frequencies where the allocated nodes may run. If there is no CPU frequency to allow the future power consumption of the cluster to be less than the power budget then the job remains pending. In the opposite case, the job is executed at the maximum CPU frequency that allows the job to be executed keeping the cluster in the power budget.

Algorithm 2: The selection of switch-offs algorithm to control the nodes switch-off reservations.

Input: The user creates a powercap reservation, indicating the interval time and the value of the powercap (P).

if $P < N.P_{min}$ **then**

  $N_{dvfs} = \frac{P - N \times P_{off}}{P_{min} - P_{off}}$

  $N_{off} = N - N_{dvfs}$

  Make a switch-off reservation of N_{off} nodes during the powercap.

else

  $\rho = 1 - \frac{1}{deg_{min}} - \frac{P_{max} - P_{dvfs}}{P_{max} - P_{off}}$

  if $\rho <= 0$ **then**

    $N_{off} = \frac{P - N \times P_{max}}{P_{off} - P_{max}}$

    Make a switch-off reservation of N_{off} nodes during the powercap.

  end

end

One of the important parts of the algorithm is the selection of the CPU frequency. Selecting a value close to the maximum will make the power consumption of the cluster to increase faster (some nodes will have to be kept idle) producing starvation of following jobs and dropping the utilization of the system. Considering that jobs may run at a lower CPU frequency (which means that nodes will consume less power) gives us extra flexibility for scheduling more jobs. However, since only one job is treated at a time and we cannot know how many jobs will follow, we need to select the best possible value of CPU frequency whenever the powercapping is activated. The optimal CPU frequency is the maximum allowed frequency that all idle nodes could run such that the total power consumption of the cluster remain within the powercap value. Since the number of idle nodes may change, the optimal CPU frequency may also change from one job to another.

The adequate energy saving mechanism is chosen in the selection of switch-offs step. System administrators can force one or another mechanism. We defined three policies *SHUT*, *DVFS* and *MIX*. *SHUT* means that the system will have the possibility

Algorithm 3: Simplified algorgithm of second part (check of powercap and choose DVFS values of jobs).

Input: The job trying to be scheduled.
job.DVFS = highest possible DVFS
while $currentPower + N_{job.DVFS} \times job.requiredNodes > P$ **do**
  if $job.DVFS == minimumDVFSpossible$ **then**
    return *Impossible to schedule the job now.*
  end
  job.DVFS = a slower value of job.DVFS
end

to switch-off nodes and keep others in an idle state if needed. *DVFS* policy means that the system will have the possibility to oblige jobs to be executed at lower CPU frequencies. Finally, *MIX* is a mixed *DVFS* and *SHUT* strategy, which considers both possibilities of saving power. In *DVFS*, *SHUT* or *MIX* modes, the system will decide which mechanism is the most suitable one using the equations introduced in Section 4.2.1.

If the powercap value is set for *now* then there could be a problematic scenario where the cluster is currently above the powercap. In this case, by default, no extreme actions are taken with the running jobs. This means that no additional jobs will be scheduled and the scheduler will wait until some jobs are completed to eventually have the power consumption of the cluster dropped to a value lower than the powercap. However, we argue that the above default way may not be accepted by some sites that may want to have extreme actions when the powercap value is set. In this case, the necessary number of jobs will be killed until the power consumption of the cluster drops instantaneously.

4.4 Implementation upon SLURM

The above scheduling algorithm has been implemented upon the open-source resource and job management system SLURM [Yoo+03]. As of the June 2014 Top 500 computer list [1], SLURM was performing workload management on six of the ten most powerful computers in the world including the number 1 system, Tianhe-2 with 3,120,000 computing cores.

In a nutshell, SLURM is designed as a client-server distributed application: a centralized server daemon, also known as the controller, communicates with a client daemon running on each computing node. Users can request the controller for resources to execute interactive or batch applications, referred as jobs. The controller dispatches the jobs on the available resources, whether full nodes or partial nodes, according to a configurable set of rules.

The power awareness of SLURM has recently been enhanced by introducing the capability to regularly capture the instantaneous consumed power of nodes, as

[1] `http://www.top500.org/list/2014/06/`

presented in Section 3.4. Coupled with the introduction of a speed scaling logic, enabling to modify the frequencies of the cores involved during the execution, this new feature helps to identify the behavior of applications in terms of power consumption when varying the frequency.

To achieve the targeted goal of powercapping, a new parameter called PowerCap is added to the controller's set of states. It represents the allowed power budget of the cluster in watts. Also, SLURM reservation characteristics have been extended by a new Watts parameter in order to specify a particular amount of power reserved for a specific time slot.

To compute the maximum power amount required to operate a cluster, new parameters are associated to the compute nodes to provide the different maximum amounts of watts consumed. Thus, IdleWatts, MaxWatts, DownWatts will respectively correspond to the amounts of watts required to operate a node in idle, fully used and down states. The down state corresponds to the state the controller uses to characterize a node not being currently accessible within SLURM (e.g. in the case of node shutdown). Furthermore, other parameters such as CpuFreqXWatts may be used to characterize a node that its CPUs runs at a specific X Frequency. While computing the instantaneous maximum amount of power of the cluster, the controller will use the known states of the nodes in order to sum up the individual maximum amounts of watts and produce a global power value for the whole cluster.

The choice of powercap scheduling mode (*SHUT*, *DVFS* or *MIX*) has been implemented as a configurable SchedulerParameter option and can be dynamically altered by the administrator without restarting SLURM services. The first part of the scheduling algorithm is triggered only in the case of powercap reservations and has the ability to reserve the shutdown of nodes. In our context, the goal is to regroup the shutdown of contiguous nodes in order to benefit of power bonus possibilities as described in Section 4.2. Hence, we coupled this feature in the first scheduling part and the shutdown of nodes is triggered through a specific type of reservations in SLURM.

The check of the powercap and the choice of DVFS values are implemented in the central part of SLURM scheduling mechanism upon the controller. When evaluating the impact of the start of a pending job, the controller will temporarily alter the states of the candidate nodes, compute the resultant consumption and compare it to the defined and planned powercap. In case of *DVFS* or *MIX* scheduling mode, the evaluated job is controlled for all different CPU-Frequencies and it stays pending only if the estimated power consumption with the lower permitted CPU Frequency is larger than the power envelope it may use. The target CPU-Frequency is selected based on the Algorithm 3. Since the *DVFS* is actually altered by the controller during the scheduling phase, the walltime of the job needs to be adapted respectively. Based on similar studies [Eti+10a], we consider that the walltime should be increased up to 60% for the minimum CPU Frequency, while intermediate values of walltimes are linearly interpolated. Note that the current code does not

make any difference in power requirements whether nodes are fully or partially used. The evaluation of new jobs only filling partially used nodes will always pass the powercapping criteria as no extra power will be required. As a result, the scheduler will tend to fill the compute nodes up to the targeted power budget.

4.5 Adapting powercapping logic for Curie

The activation of the adaptive power control of SLURM for a certain infrastructure requires an initial configuration where the maximum power consumptions of each implicated components, along with other important parameters are defined. In this section we present the study made for the adaptation of SLURM powercapping logic for Curie supercomputer.

4.5.1 Details on Curie

Curie[2] is owned by GENCI[3], it is the first french Tier-0 system opened to scientists through the participation into the PRACE[4] research infrastructure. Since its upgrade in February 2012, Curie consists of 280 Bullx B chassis housing 5,040 Bullx B510 nodes, each with two 8-core Intel Sandy Bridge processors for a total of 80640 cores. Curie was ranked 26th among the 500 most powerful supercomputers on June's 2014 Top500 list[5].

Configuration details and workload traces of Curie have been extracted at some points of its early lifetime and are used in this study to specify hardware characteristics and evaluate the behavior of the proposed mechanisms.

We have seen in Section 4.2.2 that the architecture of an HPC cluster plays an important role when considering which nodes to switch-off in order to maximize the 'power bonus'. In Curie, we distinguish 4 hierarchical levels that can be switched-off. Table 4.1 gives the power consumption of each level.

- The first one is the node level. A node is composed of 2 sockets and each socket contains 8 processors. When a node is powered-off the BMC is kept active so that the node can be powered back on through the network. This explains the consumption of 14W in table 4.1 when a node is down.
- The second level is the chassis level. Each Chassis contains 18 nodes and the power bonus is composed by global cooling fans, network switches installations such as Ethernet and Infiniband switches, optical cables, network ports. These hardware components consume an extra amount of power of 248 Watts with a power bonus of 500 Watts as we see in 4.1.
- Rack level is the third one. It is composed of 5 chassis, which contain fans and a cold door related to the liquid cooling equipment. The power consumption of

[2] http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
[3] http://www.genci.fr/en/our-computers
[4] http://www.prace-project.eu/
[5] http://www.top500.org

Level	Power consumption	Power bonus	Acummulated Power
Node (down)	14 W	-	-
Node (max)	358 W	-	344 W
Chassis	248 W	248+18*14=	344*18+500=
(18 nodes)		**500 W**	**6692 W**
Rack	900 W	900+500*5=	6692*5+900=
(5 chassis)		**3400 W**	**34360 W**

Table 4.1. Power consumption and the possible saved watts when various levels of the cluster are switched-off.

these components is 900 Watts and the bonus when switching off a complete rack will be 3400 Watts.

- Finally, the last level is the whole cluster, which is composed of 56 racks.

The power bonus values of table 4.1 imply that if we switch off a complete chassis this will allow us to completely use at least 1 extra node where-as if we switch-off a complete rack this will allow us to use at least 9 extra nodes. For example, if our model needs a power reduction of 6600 Watts and we consider that when a node is powered-on it will be used with the maximum consumption of 358 Watts then we need to switch-off 20 single nodes to reduce the power for 6880 Watts (=344*20). However if we make sure that we power off the 18 nodes of a complete chassis this will allow us to make use of the bonus and reduce the power for 6692 Watts which is even more than what we actually need. This allows us to use 2 extra nodes. In order to take advantage of the power bonus and keep more nodes powered-on, we need to prepare an efficient grouping of nodes to switch-off. Hence, that is why the choice of which nodes will be switched off takes place during the first part of our algorithm.

4.5.2 Control and Measure power

The power consumption of the different states of a node may be given by the constructor or calculated through experimentations. In our case we perform experiments of various known benchmarks using the exact same models of computing nodes and hardware components that are used on Curie.

We have run three different widely used benchmarks and one application to measure the characteristics of Curie cluster. We have chosen a first benchmark to stress all computing resources (Linpack [Lin]), a second one targeting memory (Stream [McC95]), one focused on network (IMB [Imb]) and the last one is a widely used application for molecular dynamics simulation (GROMACS [Ber+95]). Measurements have been done through the IPMI[6] interface of SLURM power profiling

[6] IPMI (Intelligent Platform Management Interface)

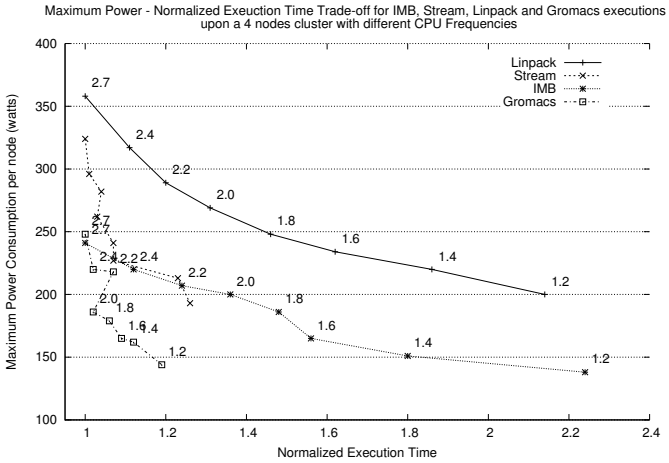


Figure 4.3. Maximum Power - Execution Time Tradeoffs for Linpack, Stream, IMB and Gromacs benchmarks at different CPU frequencies

Node state	Maximum power consumption
Switch-off	14 W
Idle	117 W
DVFS 1.2 GHz	193 W
DVFS 1.4 GHz	213 W
DVFS 1.6 GHz	234 W
DVFS 1.8 GHz	248 W
DVFS 2.0 GHz	269 W
DVFS 2.2 GHz	289 W
DVFS 2.4 GHz	317 W
DVFS 2.7 GHz	358 W

Table 4.2. Table of the maximum power consumption of a Curie node in different states.

mechanisms which have been shown to provide precise results for the consumption at the node level [Geo+c]. DVFS is a way to obtain a trade-off between power and completion time. It has been widely studied in the literature (see Section 4.1). Figure 4.3 gives the evolution of the completion time and the maximum watts consumed at different DVFS values.

Table 4.2 presents the maximum power consumption observed on a node for each DVFS value based upon the results of all 4 applications. We have also added the observed power consumption of switched-off and idle states. These measurements set the maximum Watts that a node can consume. There is huge gap between switched-off and idle nodes. Both of them do not produce computational power but a switched-off node consumes one order of magnitude less power.

Also, from these benchmarks we compute the performance impact of DVFS. For simplicity, only the maximum and minimum DVFS frequencies are taken into

Benchmark	deg_{min}	ρ	Best mechanism
NA	2.27	0	-
linpack	2.14	-0.027	Switch-off
IMB	2.13	-0.029	Switch-off
SPEC Float [Fre+07]	1.89	-0.088	Switch-off
SPEC Integer [Fre+07]	1.74	-0.134	Switch-off
Common value [Eti+10a]	1.63	-0.174	Switch-off
NAS suite [Fre+07]	1.5	-0.225	Switch-off
STREAM	1.26	-0.350	Switch-off
GROMACS	1.16	-0.422	Switch-off

Table 4.3. Comparison between DVFS and switch-off in Curie for various benchmarks.

account. Thus, in the following the *degradation* of performance is between the maximum and the minimum DVFS values. As seen in Section 4.1, the *degradation* of performances has already been studied. In [Fre+07], the authors measured the *degradation* for the NAS benchmark, the SPEC float and integer benchmarks. A *degradation* of 163% is assumed to be a good approximation [Eti+10a]. Figure 4.3 summarizes the data obtained on Curie for various benchmarks. Common values of *degradation* clearly show that shutdown is the best mechanism to use, at least for reasonable values of powercap.

In our context, the *SHUT* policy appears to be the best one. Hence, the selection of switch-offs algorithm would never mix *DVFS* and *SHUT* modes. However, we observe in the benchmarks' results that, unlike the power/performance trade-off, the energy/performance trade-off is not monotonic. The most optimal points are between 2.7 GHz and 2.0 GHz. As a consequence, we consider the *MIX* mode with higher DVFS values. The aim of this *MIX* mode is to improve performance and energy consumption while remaining under the power budget. This algorithm is the same as the one previously described but the minimum DVFS frequency is 2.0 GHz instead of 1.2 GHz. Both mechanisms should be used together when the powercap is inferior to 75% of the maximum power.In the remainder of the chapter all references to *MIX* policy consider always the high DVFS values (2.0-2.7GHz).

In case we cannot switch-off nodes, the *SHUT* mode can be implemented by keeping nodes idle. In this case, ρ becomes positive for all *degradation* values of benchmarks. Thus, *DVFS* turns out to be the best policy in all cases.

4.6 Experimental Evaluations

Our choices for experimental evaluations were to: i) use the real workload trace of Curie for approximating production executions of a large-scale supercomputer,

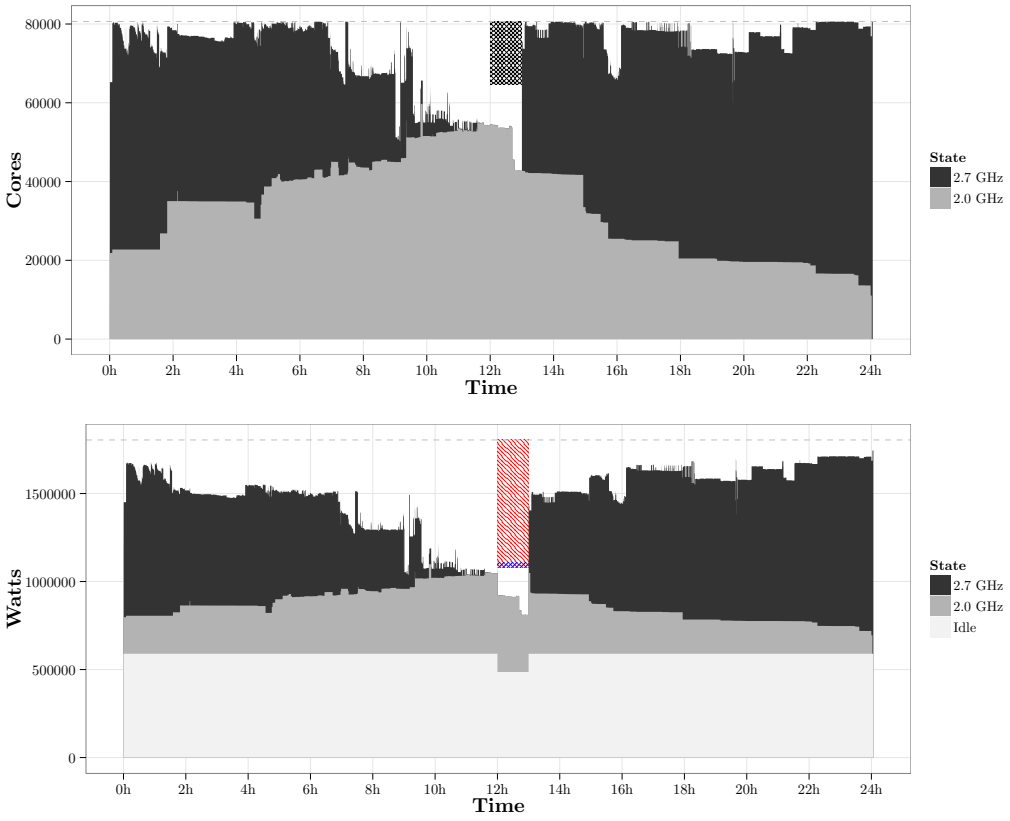


Figure 4.4. System utilization for the *MIX* policy in terms of cores (top) and power (bottom) during the 24 hours workload with a reservation (hatched area) of 1 hour of 40% of total power. Cores switched-off represented by a dark-grey hatched area.

ii) take into account the real power consumption data of Curie as discussed in Section 4.2 and iii) make use of an emulation technique to study SLURM by using only a small fraction of physical machines.

4.6.1 Platform and Testbed

The experiments have been performed upon Nova2 platform which is an internal BULL cluster dedicated for experimentations. The cluster is composed by Intel Sandy Bridge processors with 65 GB of Memory and Infiniband network.

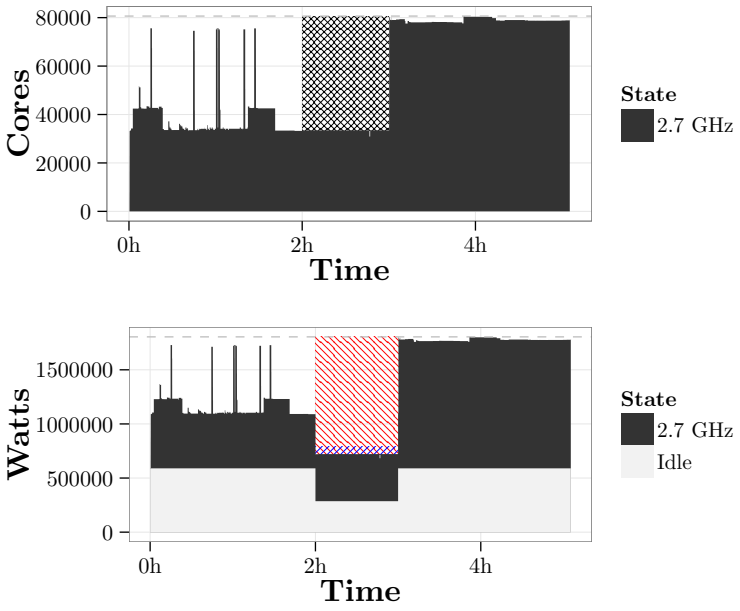
In order to enable real-scale experiments of Curie's workloads with SLURM we need to deploy a configuration of the same size. This is done by making use of an internal SLURM emulation technique called *multiple-slurmd*. This technique is described and validated in [GH12]. In our context, we deploy 5040 nodes of Curie with only 16 physical nodes of our experimental platform Nova2.

4.6.2 Methodology

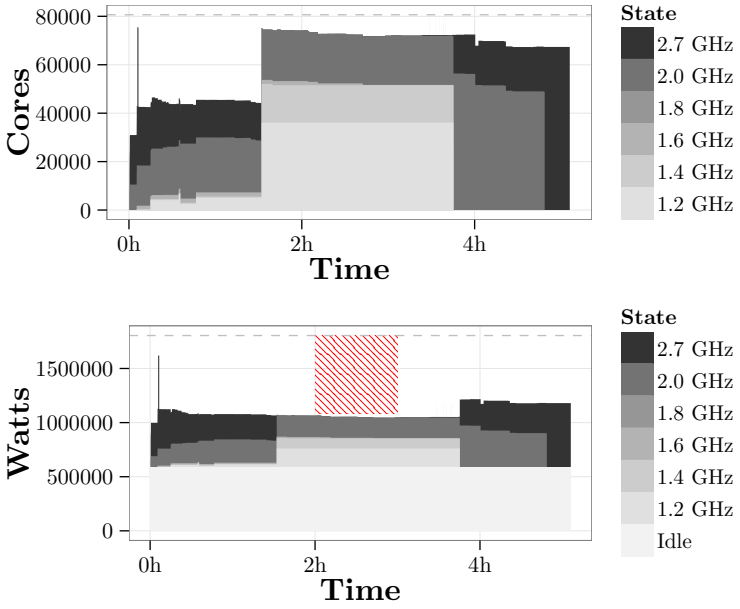
We propose to replay time intervals extracted from a real workload trace of Curie supercomputer in 2012[7]. In more detail, we select three intervals of 5 hours and one interval of 24 hours with high utilization (most cores are used for computations), big number of jobs in the queue and short inter-arrival time. The intervals used in the following experiments are: i) *medianjob*, with jobs that are representative of the whole workload, ii) *smalljob*, with more small jobs than in the *medianjob* interval, iii) *bigjob*, with more big jobs than in the *medianjon* interval, iv) *24h*, with jobs that are representative of the whole workload. In the extracted traces, Curie is overloaded: there are always at least enough jobs in the submission queues to fill a second cluster of the same size. Most of the jobs are small compared to Curie size, 69% are jobs that need less than 512 cores and ran for less than 2 minutes. 0.1% of jobs are huge, these jobs use more than the equivalent of the whole cluster for 1 hour. It is important to note that in these particular traces, users estimate runtimes badly. In average, they request about 12670 times more walltime than needed (median: 12000). This leads to difficulties for the system to schedule correctly jobs [Tsa+05].

As we are only interested in the RJMS internal decisions, hence the jobs are replaced by simple "sleep" commands. On the original workload, all the jobs were run at maximum DVFS. If our powercap scheduling algorithm decides to run a job at a lower speed, the emulated job will be executed slower. We choose to use a performance *degradation* of 1.63 (1.29 with *MIX*) for all jobs, as our experiments and related works agree that it is a reasonable value (see Section 4.5.2). This performance degradation is computed with the maximum speed (2.7 GHz) and minimum speed (1.2 GHz or 2.0 GHz with *MIX*), the intermediate values have been linearly interpolated.

[7] `http://www.cs.huji.ac.il/labs/parallel/workload/l_cea_curie/index.html`



(a) Powercap of 60% with mainly big jobs and *SHUT* policy



(b) Powercap of 40% with mainly small jobs and *DVFS* policy

Figure 4.5. System utilization for the diferent runs in terms of cores (top) and power (bottom) during 5 hours workload with a powercap reservation (hatched area) of 1 hour of 60% or 40% of total power. Cores belonging to switched-off nodes are in cross hatched area.

The replay of the time interval is based on the four following phases: i) the environment setup: SLURM is set in the closest state of the original run. We put in place the original SLURM configuration of Curie and modified only the parameters that allow our replay (node names, characteristics and power values), ii) the interval initial state setup: runtime characteristics are put in place (queued and running jobs, fairshare values for each user), iii) the actual workload replay: jobs are submitted with the same characteristics as they were on the original run (simple 'sleep' jobs, not real executions), iv) data post-treatment: once the replay of the time interval end, we stop SLURM and then collect and gather information about the replay by the end of the interval (jobs state, outputs and characteristics).

This methodology has some limitations. The initial placement of jobs is not always respected, and we do not replay node failures. Furthermore, job submissions depend of the response time experienced by users [SF07]. These limitations imply that comparisons to the original traces can not be conducted, but, as the replay is deterministic, we can compare the different replays.

In the experiments reported in the next section, we are evaluating the three different policies *DVFS*, *SHUT* and *MIX*. The policies are tested under three powercap scenarios reserving respectively 80%, 60% and 40% of the available power budget for one hour in the middle of the replayed interval. Powercap reservations are made in the beginning of the workload replay. All experimental results are compared between them along with a simple run where no powercapping takes place. Our goal is to compare system utilization in terms of CPUs and power usage along with the effective work for each scenario.

4.6.3 Analysis of results

Figure 4.4 shows the system utilization (top) and power consumption (bottom) during the replay of the *24h* workload using the *MIX* policy. The grey area in the top figure represents the system utilization of jobs executed upon cores with CPU Frequency of 2.0 GHz whereas the black area represents those running with 2.7 GHz. In the bottom figure the light grey area represents the minimum power consumption of the system if all nodes are idle and no jobs are executed, the grey area represents the additional power consumption of jobs whom the cores compute with 2.0 GHz and the black area reflects the additional power consumed by jobs that compute with 2.7 GHz. The reserved power, is represented by the hatched area in the bottom figure, thus the allowed powercap budget, for that period, is the remaining power below that area. The powercap is triggered in the beginning of workload that is why we observe that jobs are launched with lower CPU frequency directly from the start. Since we are in a *MIX* policy, the first part of the scheduling has reserved a certain number of nodes to shutdown. This can be viewed by the cross hatched area in the top figure during the period of powercap. The small blue cross hatched rectangle below the powercap reservation rectangle represents the bonus power gained back by the grouped shutdown of continuous nodes. This is actually gained

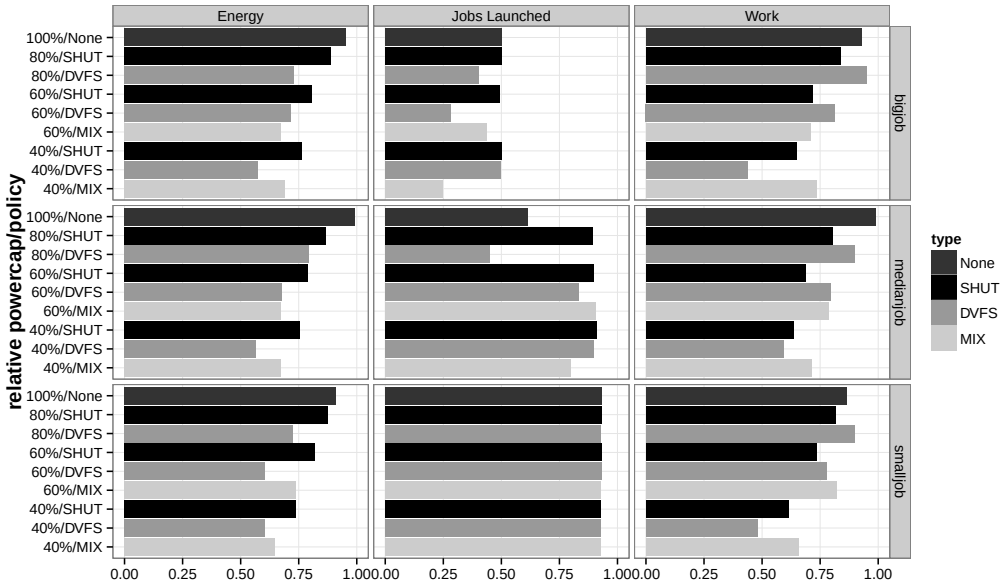


Figure 4.6. Comparison of different scenarios of policies and powercaps based on normalized values of total consumed energy, launched jobs and accumulated cpu time during the 5 hours workload interval.

power being used by the system for computations but it is plotted upon the reserved power hatched area to better reflect the proportions between them.

It is interesting to observe how while approaching the powercap reservation the system prepares itself for limited power usage by launching more jobs with 2.0 GHz. Similarly after the powercap has passed, utilization of 2.7 GHz cores increases because new jobs are launched with maximum frequency, while older jobs launched with 2.0 GHz, launched before or during the powercap, still remain but gradually decrease. After the powercapped period, we see that the system utilization in terms of cores increases directly to nearly 100%. It seems that a large job allocating more than 40000 cores was scheduled directly after the powercap period. This large job seems to be blocking other smaller jobs that follow and backfilling does not seem to work since thick gaps are witnessed during the powercap interval. Based on previous observations; backfilling is not efficient because of wrong walltime estimations.

If we take a look at other *24h* runs with a powercap of 40%, *DVFS* and *MIX* show similar results: a work around 85% of the total possible work, while *SHUT* has a work of 94% of the total possible work. It is interesting to note that the energy consumption is a the lowest in the *MIX* mode.

Figures 4.5a and 4.5b represent the system utilization for the *smalljob* and *bigjob* workloads with different use cases. They are based on the same representations as the previous figure. The difference is that the left one provides results with *SHUT* policy whereas the right one with *DVFS* policy for 60% and 40% powercaps respectively. In the left figure we can observe how the shutdown of nodes makes big space in order to adapt the workload without wasting un-utilized cores. In

addition, we can see the value of power bonus due to the regrouping of nodes to be switched-off. Without the first part of the scheduler this bonus would not be possible. Furthermore, we see how the system utilization increases directly after the powercap interval to 100%. It is interesting to see how backfilling does not take place a lot while preparing for the powercap period. This is due to the nature of type of jobs which is mainly big jobs along with the walltime problems that we explained before. In the right figure, different tones of grey represent the different frequencies until the black area which is 2.7GHz. We can see how the appearance of low CPU Frequencies increase while approaching to the powercap period with a total disappearance of 2.7GHz frequencies in close regions to the powercap interval. *DVFS* policy manages to obtain high system utilization with a low power consumption.

We also have done several run with DVFS and switch-off mechanisms deactivated. The only solution for our algorithm is to let nodes idle. As expected, this solution has the worst work (about 40% lower than other modes), while keeping about the same energy consumption.

Let us now look at the impact of the policies for the performances. Figure 4.6 provides the different runs executed to compare the performance of the different powercap scheduling modes for 5 hours workload. Considering columns we observe the total consumed energy, the number of launched jobs and the total work. In terms of rows we have different groups. The groups based on the workload (left): *bigjob*, *medianjob* and *smalljob* along with the groups representing powercap reservations: 100%, 80%, 60% and 40% which reflect the system power which is allocated for computation. Furthermore distinctions between the different scheduling modes is also made in groups of particular rows in the figure. Only jobs that were running during the replayed time interval are taken into account, and all measures are normalized to the maximal possible value. In the histograms we observe that *DVFS* mode's work is always larger than *SHUT* mode's work and that is because jobs run with lower CPU Frequency and hence the walltime is increased. The *MIX* mode provides most of the time the best energy consumption, while having a work in the same order of others modes.

In the *medianjob* workload, 100%/None and 80%/DVFS runs launched less jobs than others run, while having a high utilization. It seems that in these runs, the algorithm chooses to schedule a huge job preventing a large number of other jobs to be launched.

If we take a look at each mode independently we can see that for every type of workload work and energy decrease proportionally to the powercap diminution. Furthermore, *DVFS* mode seems to be decreasing more rapidly below 60% whereas *SHUT* and *MIX* modes appear to be more consistent. Switch-off mechanism (*SHUT*, *MIX*) seems to be more efficient if we consider the tradeoffs energy/work and this is basically related to the in-advance preparation in the first part and the gained power due to the bonus.

4.7 Conclusions

We presented in this chapter a new scheduling algorithm for dealing with power limitations in large scale HPC clusters. The algorithm was developed for a resource and job management system and implemented upon SLURM. It is composed of two phases: an first part where the planning takes place (choice of policy, selection of group of nodes to be switched-off, etc.) followed by an second part where the power reduction is applied. The implementation upon SLURM resulted into the design of three powercap policies, namely *DVFS*, *SHUT* and *MIX* which respectively take advantage of CPU Frequency scaling, nodes shut down and mixed capabilities in order to achieve power reductions whenever needed. One of our main objectives was to enable the scheduler to determine automatically the best powercap mechanism for the nodes and we showed how this depends on the architecture, the power consumption of the components and the actual workload. The new developments are on the main branch of SLURM in the upcoming version 15.08. As far as our knowledge, this is the first work that considers powercapping techniques in the level of job scheduling for a resource and job management system in HPC. The study allowed us to validate the algorithms and evaluate the different policies through real-scale emulation of a petaflopic supercomputer. In particular we performed experiments with emulation of Curie's characteristics and calculated power values using replay of a real workload trace collected from the production of Curie on 2012. The experimental results validate the model and provide interesting initial insights. Switching-off nodes appear to be the most efficient policy in our use cases of less than 60% powercaps, mixed policy seems to be the more consistent one and finally frequency scaling provides better results with large powercaps of 80%.

5 Budget control with energy-aware resource management and job scheduling

> *When Thomas Edison worked late into the night on the electric light, he had to do it by gas lamp or candle. I'm sure it made the work seem that much more urgent.*
>
> **— George Carlin**

To control the energy consumption of huge HPC platforms, multiple techniques have been developed. One of them, the powercapping (presented in the previous chapter) limits the power consumption to a certain threshold. Most methods control the energy consumption through the control of the instant power consumption. While these techniques have shown their effectiveness, they lack adaptability that can be given by only controlling power.

The presented techniques are similar to a power adaptive scheduling except that we limit execution under maximum energy consumption for a time duration. The developed techniques are an extension of backfilling. It keeps the idea that a job can run before a job of higher priority only if does not delay it. Instead of only considering the availability of computing resources, it also takes into account the availability of energy. Overall, it enables the platform to meet a certain energy consumption budget goals. In addition, it also supports opportunistic shutdown of nodes in order to reduce the energy consumption. Through intensive simulations, we show that our techniques keep high performances while staying under the energy budget.

The final goal of this study is to make an algorithm that can dynamically adapt to the electricity price, and thus reduce the operating cost of supercomputers. In this chapter, we present the first step: an adaptation of a standard scheduling algorithm in order to limit the energy consumption during a time period. This preliminary study has been done in collaboration with Pierre-François Dutot, Yiannis Georgiou, Laurent Lefevre, Millian Poquet and, Issam Rais, and it is partially supported by the ANR project MOEBUS.

5.1 Problem description

We are interested in the problem of scheduling jobs on a large number of resources, with the following constraint: during certain time frames, the energy consumption of the whole computing center cannot exceed a given limit. The energy

limit we are using here (in joules) must be distinguished from an electrical power limit (in watts).

In order to reduce and control energy consumption, there exist techniques that allow energy savings from different levels. Some of these techniques are introduced by architecture manufacturers (ex: DVFS), others are invariant possibilities of the infrastructure (ex: node On/Off, heterogeneity, etc). Most of these techniques are available at the user level, making it complicated for an energy aware user to properly use them. The aim is to hide the usage of such techniques to the final user, while benefiting the possible energy savings made by a wise usage of such techniques. In this chapter, we lean on a On/Off technique that could greatly impact the overall energy consumption.

DVFS can be used to control the power of a given job. However, previous studies [Geo+b; Geo+15] have shown that the control of the energy consumption is not obvious. Depending on the job, a given DVFS value may increase or decrease the energy consumption. The energy consumption using various DVFS values should be carefully studied for each job to be used wisely. This is why we choose to do not take into account DVFS in this study.

5.2 Related Work

5.2.1 Controlling power and energy consumption

Many papers focus on controlling the power consumption [Sar+14; Rou+12b; Eti+12a]. 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. Powercap mechanisms have two major drawbacks. It may require a high knowledge of the running applications (to tune DVFS or a similar technique). It also delays some jobs. In our study [Geo+b], we found that only controlling the power increases the turnaround time of big jobs (as it is harder for them to "fit" in the powercap). This is why we focus here on energy budgeting, we want to keep the benefit of controlling the cost of the cluster while not discriminating some type of jobs.

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 of this field because they are applied to real-time small scale systems.

In [MV15], Murali et al. study a metascheduler that control 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 the scheduling problem with 2 periods: one where there is a limit in energy and one with no limit. While the approach is interesting, the algorithm that they use is not scalable and is hardly extendable with other constraints. In a study. [Khe+14],

Khemka et al. maximize a "utility" function in a daily energy limited cluster. They resolve the problem with an offline heuristic. We do not rely on utility function, instead we use classic scheduling objectives as described in Section 2.4.

5.2.2 Opportunistic shutdown

The energy consumption of a node is decomposed into two components: the static power (P_{idle}) and the dynamic power (P_{dyn}). P_{idle} is the power used when no activity is witnessed on a node. It represents the majority of the overall energy consumption of a computing node. P_{dyn} represents the consumption of a node when there is some activity through logic gates inside the CPU [BH07].

P_{idle} is not negligible, however the energy consumption of shutted-down node is very small [BH07]. There exists a simple solution to take into account most of this feature: opportunistic shutdown. This technique consists in shutting down nodes that are idle. The nodes are monitored, when a defined idle period is witnessed, the decision of shutting-down the nodes is made. As shown in [Hik+08], such a solution could lead to non negligible energy savings.

When a very low budget is used, a lot of nodes won't be able to run jobs. Thus, these unused nodes will consume P_{idle}. If the opportunistic shutdown is enable, unused nodes will consume only a fraction of P_{idle}.

However, this solution has some limitations. One of them is the costly time and energy taken to switch off and switch on nodes. It can take several minutes while the node is consuming its maximum power [Org+08].

5.3 Proposed algorithm

5.3.1 Desired properties

Obviously, the proposed algorithm should support an energy budget during a time period. This energy budget should be strictly respected. Moreover, the algorithm should be modular enough to support extra features, like opportunistic shutdown. It should also use a well known and popular heuristic to be able to scale up and maximize its adoption.

5.3.2 Algorithm description

We choose to base our algorithm on Easy Backfilling. A description of this algorithm is summarized in Chapter 2. It has all the desired properties, but the energy ones.

In the Algorithm 4, we add in green an overview of the modifications done on Easy Backfilling in order to support energy budgets. All the modifications have been done to keep the basic idea of Easy Backfilling: a job cannot be backfilled if it delays the first job that cannot be started immediately.

The first step is to define a $powerlimit$ which is equal to the budget divided by the length of the budget period. It is like we spread the energy budget uniformly over the time period. Backfilled jobs cannot be started if it will make the power

Algorithm 4: Energy Budget Backfill algorithm

```
for job ∈ queue do
    if job fits in system and enough energy has been saved to start job then
        launch job;
        remove job from queue;
    else
        break;
    end
end
firstJob = pop first element of queue;
Make a reservation in the future for firstJob;
Make an energy reservation in the future for firstJob;
for job ∈ queue do
    if job fits in system and enough energy has been saved to start job then
        launch job;
        remove job from queue;
    end
end
Remove reservation for firstJob;
Push back firstJob at the top of queue;
```

consumed exceed the $powerlimit$. Other jobs can exceed this limit if enough energy has been saved.

With this algorithm, if the energy budget is unbounded, it will produce the same scheduling as EasyBackfilling. Also, if the energy budget is very low, we expect our mechanism to start jobs in order of the queue.

In Algorithm 4, the mechanism to determine how much energy has been saved is not described. Every monitoring period we accumulate the energy consumed on the cluster since the last monitoring and the difference with the $powerlimit$ is accumulated in a variable. This variable holds how much energy has been saved (this variable may be negative if the cluster consume too much energy).

5.3.3 How to not delay first job reservation?

When a job cannot be started right now and has to be delayed (this job is called $firstJob$ in Algorithm 4), it has to reserve enough energy to run as soon as possible. How to make this energy reservation?

We develop two different mechanisms. The first one, $reducePC$, reduces the $powerlimit$ when backfilling jobs by the amount of energy that we have to save. Its simplicity is its main strength, and any system that already integrates a powercapping algorithm can implement this variation very easily.

The second one, $energyBud$, reserves some energy and forbids backfilled jobs to consume this amount of energy. This one should perform better as it takes a deeper look on where energy may be saved.

The main difference between the two mechanisms can be seen when a short job that uses a lot of processors is being backfilled. With $energBud$, if enough energy is available it will be launched. However, with $reducePC$, as the $powerlimit$ has been reduced, the job will not be started.

5.3.4 Power and energy consumed by a job

A drawback of our mechanism is that we should know how much power and energy a job will consume. Fortunately, a fixed value for power consumption is enough to make our mechanism work. Let us explain why. The real energy consumption is obtained on monitoring phases. If the fixed value is higher than the real power consumption, the mechanism will accumulate more energy and thus, backfill more jobs. However, if the value is lower than the actual value, fewer jobs will be backfilled. Previous studies have shown [Tsa+07; Gau+] that if more jobs are backfilled then we observe better performances. Thus, we recommend setting a value of the estimation higher than the $real$ one.

To obtain this value we recommend to run a CPU intensive application on the targeted cluster (like those of the Linpack suite) and use the maximum power consumption observed. In this case, the estimations are always above the actual power and energy consumption and thus, the energy budget is fulfilled. Even more, the performance degradation is limited.

5.4 Evaluation

The aim of the simulations are to answer the following questions:

- how better are we compared to a standard powercap mechanism?
- what is the gain of activating opportunistic shutdown?
- if we reduce by 80% the budget, does it reduce by 80% the performances?
- Which is the best one, $reducePC$ or $energyBud$?

5.4.1 Simulator

In order to evaluate our algorithm, we chose to analyze its behavior in simulations. Fo this purpose, we used Batsim [Dut+16], a batch scheduler simulator based on SimGrid [Cas+14]. We chose to use Batsim rather than an ad-hoc simulator for separation of concerns, to avoid implementation issues and to ensure the durability of the algorithms we propose.

In our case, we chose to compute jobs that are defined by a computation vector $c_j = \{c_{j_0}, c_{j_1}, \cdots, c_{j_n}\}$ where n is the number of requested resources for job j and where each c_{j_k} represents the amount of computation on the k^{th} resource allocated to job j. Each computational resource $m \in M$ has a set of power states P_m where each power state $p \in P_m$ has a computational power cp_p (in flop/s), a minimum electrical power consumption $ep_p^{\wedge}$ (in W) and a maximum electrical power consumption $ep_p^{\vee}$ (also in W). The computation load $l_m(t)$ of each resource m at any simulation time t is computed by Batsim and is a real number in $[0, 1]$ where 0 is an idle resource and

1 a resource at maximum computational power. Let $p_m(t)$ be the pstate in which resource m is at simulation time t. The instant electrical consumption of resource m at time t is noted $P_m(t)$ and computed as the linear interpolation between $p^{\wedge}_{p_m(t)}$ and $p^{\vee}_{p_m(t)}$ in function of $l_m(t) : P_m(t) = p^{\wedge}_{m_h(t)} + (p^{\vee}_{p_m(t)} - p^{\wedge}_{p_m(t)}) \cdot l_m(t)$ and is expressed in watts. The energy consumption of resource m during the period T is then given by $E_m = \int_T P_m(t)dt$ and is expressed in joules.

The heuristics and mechanisms described in this chapter were all integrated in the Batsim code base.

5.4.2 Calibration

To calibrate our simulator, we made various runs on the Taurus nodes of Grid5000 [Bal+13]. These nodes are Dell PowerEdge R720 nodes with 2 Intel Xeon E5-2630 per node. They are equipped with watt-meters that allow precise measures of their energy consumption.

First, we switch off and on several nodes. On every run, we retrieve the time and energy consumption for switch on and switch off sequences. Thus, we launched 50 switch on and 50 switch off sequences for every node and identify their respective consumption in time and energy. On every node, we then obtain an average for switch on and switch off sequences in time and energy.

We then run a Linpack benchmark on these nodes. We used the average power consumption to calibrate the simulator. The maximum power consumption is used within the algorithm (as recommended in Section 5.3.4).

5.4.3 Testset

We use the simulator to assess a wide range of situations.

First, we select 5 different weeks in 3 workload traces available on the Parallel Workload Archive [Fei+14]. These 3 workloads corresponds to different cluster of different sizes:

- Curie, 80640 processors. The trace date from 2012 and last 3 months.
- MetaCentrum, 3356 processors. The trace date from 2013 and last 6 months.
- SDSC-Blue, 1152 processors. The trace date from 2003 and last 32 months.

The selected weeks have high utilization (because scheduling decisions have more impact during these periods) and were selected randomly across the full traces. At the end of this process we have 15 different traces.

Jobs are simulated as CPU bound jobs. Incorporating a network communication model for each job is a study in itself and doesn't add much for the evaluation of our mechanisms. Still, we forced parallel jobs to be scheduled on continuous processors to mimic topology constraints.

We apply an energy budget that last 3 days in the middle of each traces. With this setting, we can observe the impact of the energy budget on the metrics during the energy budget but also before and after it. On the remainder of the chapter, we express the energy budget with a percentage. 100% corresponds to the energy that

the cluster would have consumed if all the processors on the cluster were used. We run each traces with the following value: 100%, 90%, 80%, 70%, 60%, 50%, 49%, 30%. 49% equals to the energy of the cluster where all processors are idle.

We test 4 different mechanisms. The first one is the standard Easy Backfilling. As this mechanism does not support energy budget, we only run this one when the energy budget is at 100%. The second one is the $powercap$ mechanism. In this mode, a power limitation is set during all the energy budget time period. The power limit is equal to the energy budget divided by the length of the period. Then, Easy backfilling is executed, but, no job can go above this power limit. The last two ones are the two different version of our mechanism, $energyBud$ and $reducePC$.

Also, we run each algorithm that supports power budget with and without opportunistic shutdown. This gives us 735 simulations.

5.4.4 Results

To be able to compare results from different cluster and traces, all results are given relative to Easy Backfilling relative. We then compute the average of these relative values.

How better are we compared to powercap?

Figure 5.1 depicts the relative utilization for each experimental condition. As it is the relative utilization, points on 100% do not mean that the utilization is at 100%; it means that the utilization is the same as the one obtained with the Easy Backfilling algorithm. The black line is explained in Section 5.4.4. We observe that $energyBud$ performs better than other mechanisms when the energy budget is high. For lower budget, it is $reducePc$ that performs better.

In Figure 5.2, the relative AVEbsld for each energy budget is shown. This time $energyBud$ with the opportunistic shutdown outperform all other mechanisms. Surprisingly, the $powercap$ mechanism is not the worst as it shows results comparable to $energyBud$ without opportunistic shutdown. One should not forget that the Easy Backfilling algorithm intends to maximize utilization. AVEbsld is not considered in this algorithm.

In terms of energy, Figure 5.3 presents the energy consumed during the week relative the total energy consumable during the same period. It appears clearly that when opportunistic shutdown is on, the cluster consumes less energy. Also, $energyBud$ consumes more energy than $reducePc$ which consumes more than $powecap$. Our mechanisms do not try to minimize the energy consumption. They try to keep it under a certain value. This behavior can be observed on this figure: $powercap$ has a very low energy consumption which can be seen as a non utilization of the energy saved. At the opposite, $energyBud$ is quite successful at using saved energy as it have a high utilization while having a high energy consumption.

What do we gain by activation opportunistic shutdown?

Table 5.1 shows the average improvement when activating the opportunistic shutdown on different measures. To compute this table, we take every experimental

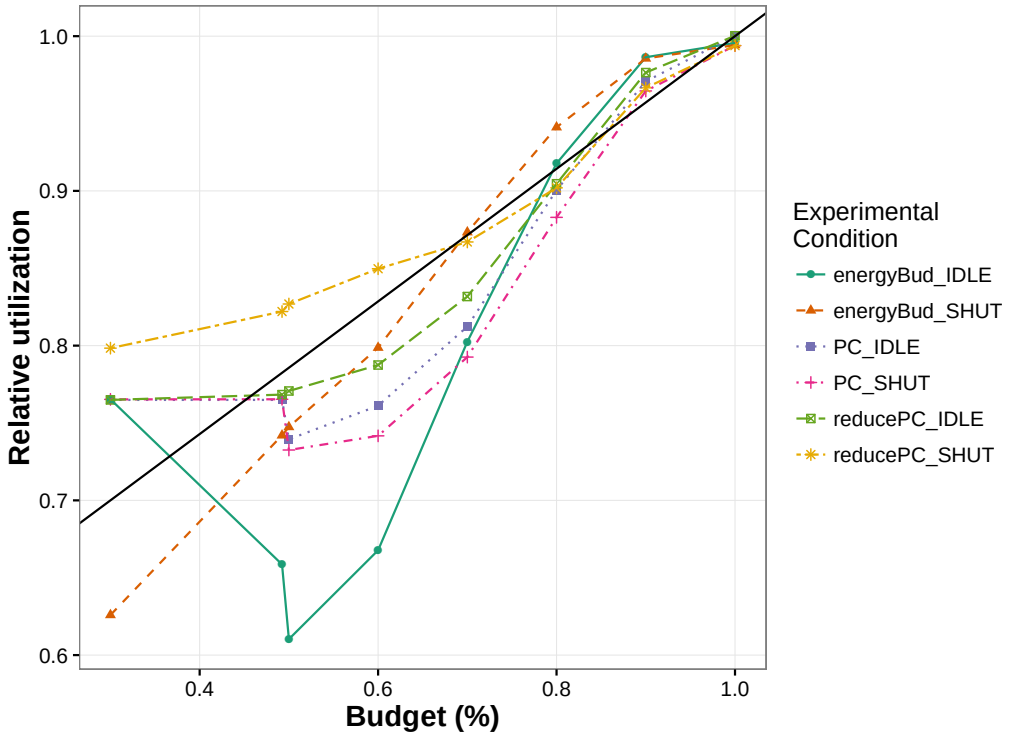


Figure 5.1. Relative system utilization during the week, Easy backfilling utilization for each traces is used as the baseline. The black line represent a theoretical performance baseline.

condition on each trace with opportunistic shutdown and compared it to the same experiment without opportunistic shutdown. This table presents the average of these computations.

As expected, by activating the opportunistic shutdown the energy consumption of $powercap$ decreases. On the two other mechanisms, it decreases less because the energy saved is used to launch more jobs. $energyBud$ takes the most of the opportunistic shutdown. While $reducPC$ and $energyBud$ increase the number of jobs started by the same amount, $energyBud$ improves far more the utilization and AVEbsld.

	Improvement		
Measure	$powercap$	$energyBud$	$reducePC$
AVEbsld	0.27 %	-9.83 %	1.91 %
Utilization	-0.37 %	2.05 %	1.22 %
Number of job started	-0.13 %	1.66 %	1.67 %
Energy consumed	-4.62 %	-1.32 %	-1.79 %

Table 5.1. Average improvements on different measures when activating shutdown.

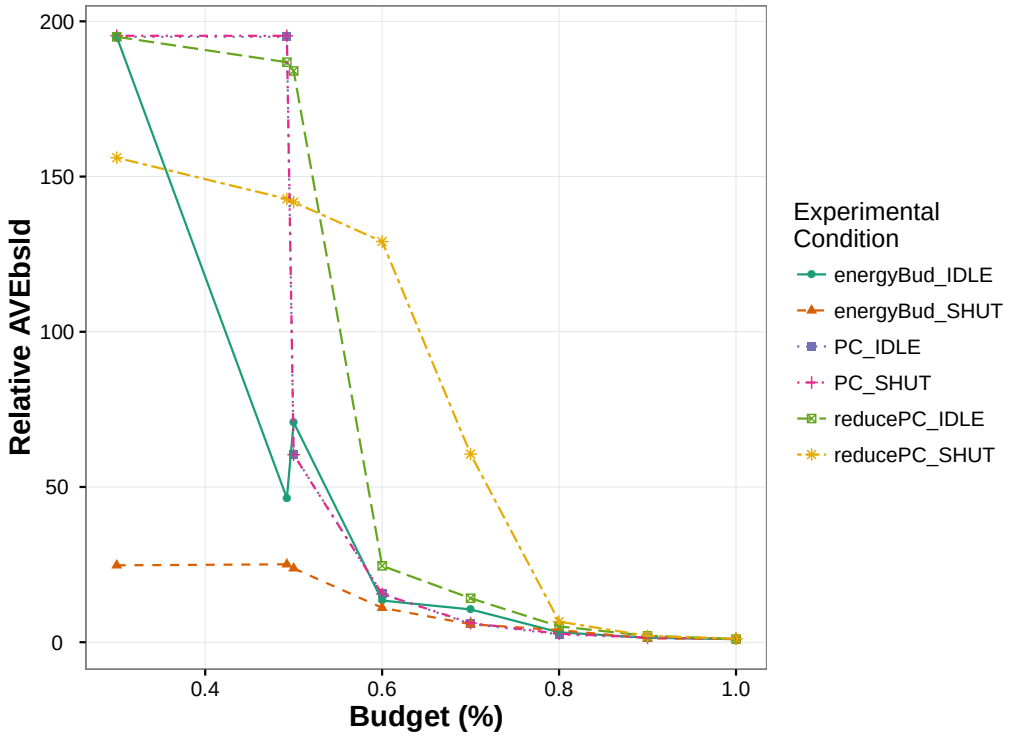


Figure 5.2. AVEbsld for all jobs relative to the AVEbsld of Easy Backfilling.

If we reduce by 80% the budget, does it reduce by 80% the performances?

In figure 5.1, the black line represents a theoretical performance baseline. If we reduce the energy budget, one can expect the performance to decrease by the same amount. This is what this line represent. The line is not the identity because the energy budget only last 3 days during the 7 days considered. Thus, this theoretical performance baseline is formulated as:

$$utilization(budget) = \frac{3}{7} \times budget + \frac{4}{7}$$

If a point is below this line, it means that the performance has decreased more than the energy budget have been decreased.

Surprisingly, for an energy budget of 90% all points are above the line. It means that, even with a simple $powercap$ mechanism, we achieve a better energy efficiency than Easy Backfilling. For $energyBud$, this is also true for an energy budget of 80% with and without opportunistic shutdown. As already observed before, $reducePC$ with opportunistic shutdown is above the line for low energy budget.

Which is the best one, reducePC or energyBud?

Depending on the objective consider and the budget, the best one change. However, we can considered that limiting the energy consumption down to 50% of the total energy consumption is a very drastic limitation. Thus, this will not happen often in real environments. This is why we consider $energyBud$ the best proposed

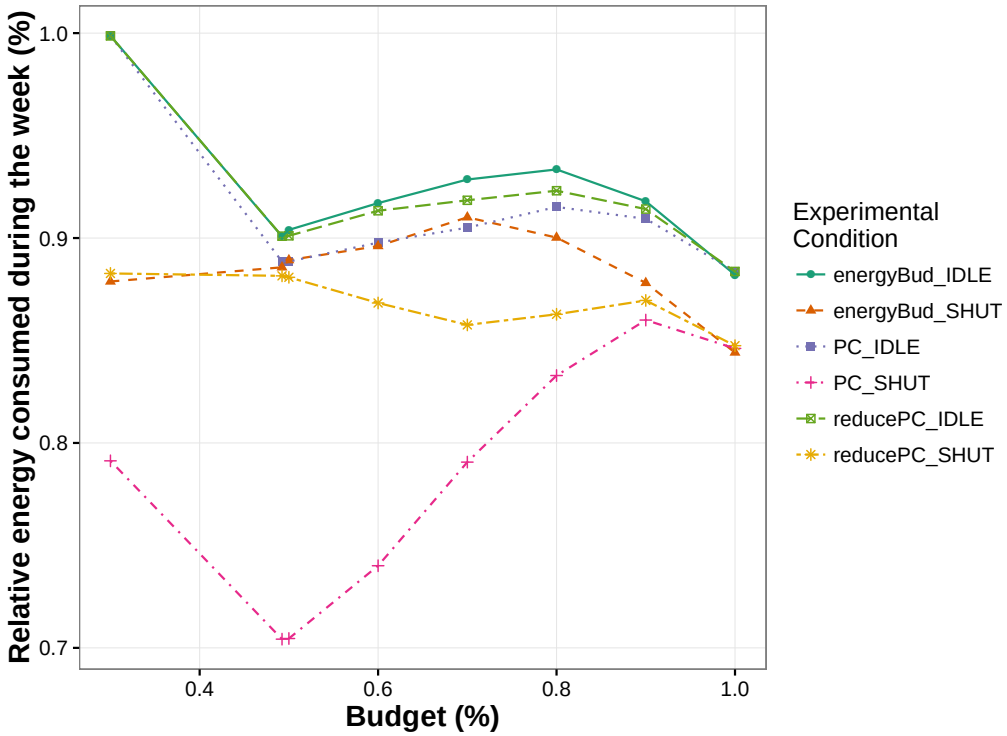


Figure 5.3. Energy consumed during the week compare to the maximum energy consumable during the same period.

mechanism. Moreover, this mechanism provides the best AVEbsld results even with very low energy budget.

5.5 Conclusion

The purpose of this study was to extend the widely used Easy Backfilling mechanism in order to support energy budget periods. We proposed two new alternatives and show their effectiveness on a wide range of simulations. These new mechanisms are not only a way to control the energy consumption of small to big clusters but also show their effectiveness on different measures. Moreover, when the energy budget available is high, our proposed mechanisms improve the energy efficiency of the cluster.

Acknowledgment

We thank gracefully the contributors of the Parallel Workloads Archive, Travis Earheart and Nancy Wilkins-Diehr (SDSC Blue), Joseph Emeras (CEA Curie), the Czech National Grid Infrastructure MetaCentrum (Metacentrum) and of course Dror Feitelson.

6 A Scheduler-Level Incentive Mechanism for Energy Efficiency in HPC

> *A skillful warrior marches her troops into battle by stirring up an overwhelming force of momentum.*
>
> **— Sun Tzu, Art of War**

Various methods have been proposed for reducing energy consumption. The design of an energy-efficient supercomputer involves for instance accelerators (GPUs or Intel Phis), which are more energy-efficient than standard CPUs for some regular workloads; or, water cooling, which disseminates heat more efficiently than air-conditioning. Yet, once a system is built, such "black-box" approaches have limited effects. For instance, by reducing the speed of processors, we reduce their voltage, and thus their power consumption. However, not only the speed-scaling slows down the computations; more importantly, the total savings are very limited. In Section 3.4, we speed-scaled processors of a cluster while measuring run-time and energy consumption of typical HPC applications, from computationally-intensive LinPack to communication-bounded Intel MPI benchmarks. Depending on the type of applications, speed-scaling increased the runtime twice; yet, electricity used was lowered by only up to 30%. Thus, we need to motivate the user to actively participate in making her computations energy efficient.

Significant savings can be made by designing energy-efficient software. The methods range from changing algorithms (e.g., by reducing communication [Dem+12]), to requesting a lower processor voltage when the application enters a non-computationally-intensive phase. The key to these savings is that only the programmer is able to precisely decide when to slow down the hardware without a large impact on the code's observed performance.

However, in a supercomputer shared by many users, there is no incentive for users to adapt energy-efficient software—and usual, minor deterrents from doing so, ranging from anxiety about performance to a "If it ain't broke, don't fix it" attitude.

We propose to shift the focus of a scheduling policy from processors to what is currently the true limit of large scale supercomputers: energy. We assess this idea by creating EnergyFairShare (EFS) scheduling algorithm. EnergyFairShare uses a well-known algorithm for sharing resources, FairShare; but the resource that is to be shared fairly is the energy budget of a supercomputer; not its processors. Consequently, users' jobs are prioritized according to their past energy consumption.

Once a particular user exceeds her assigned share of the total energy, the priority of her jobs is lowered; thus, in a loaded system, the jobs stay longer in the queue. This mechanism creates an incentive for users to save energy.

Moreover, EnergyFairShare may be used to achieve fairness in large supercomputing centers managing heterogeneous resources—from various kinds of cluster nodes (e.g., fat nodes, accelerators, FPGAs) to machines (e.g., an x86 cluster, or a BlueGene) to specialized equipment. Each resource type can be abstracted and characterized by its energy consumption. In a supercomputing center managing heterogeneous resources under a common energy budget, a user would have an incentive to choose the most energy-efficient resource for her needs.

We tested EnergyFairShare by simulation to assess the impact of jobs' energy efficiency on their queuing performance. As there are no available workloads that show jobs' energy consumption, we extended the standard workloads by assigning to each job its simulated consumption which was based on the job's size. Our results show that the energy efficient jobs have, on the average, a stretch (slowdown) lower than the stretch of the standard and gluttonous jobs.

To validate our ideas in real-world settings, we implemented EnergyFairShare as a scheduling plugin for SLURM [Yoo+03], a popular HPC resource and job management system (RJMS). Our plugin is compatible with existing FairShare policies. It obtains jobs' energy consumption data through SLURM energy accounting framework. EFS rewards energy-efficient executions with higher prioritization within the scheduler.

This chapter matches a publication [Geo+a] made in collaboration with Yiannis Georgiou, Krzysztof Rzadca, and Denis Trystram.

6.1 Related work

The main contribution is to propose to treat the energy as the limiting resource and to share the available energy budget in a fair way using the classical FairShare algorithm. Thus, below we argue that (1) in existing HPC resources, it is possible to account to a job the energy it consumes; (2) users can save energy by choosing energy-aware algorithms and libraries (thus, it makes sense to provide an incentive to save energy); (3) FairShare is a standard algorithm to achieve CPU fairness between users (many alternative approaches and algorithms exist, but some of them can be adapted to energy similarly to our approach).

6.1.1 Measuring energy consumption

In order to manage the energy budget of a supercomputer, we need to precisely measure the total energy consumed by each job. Fortunately, energy consumption is, or quickly becomes, a key issue in very diverse types of resources: from mobile devices, in which the goal is to extend the battery lifetime, to supercomputers. Thus, modern hardware provides various interfaces to monitoring energy consumption. Standards include the Intelligent Platform Management Interface (IPMI, [Int]) that

uses a specialized controller (Baseboard Management Controller, BMC) to monitor various hardware parameters of a system, including power. Intel RAPL (Running Average Power Limit, [Rot+12]) enables the operating system to access the energy consumption information for each CPU socket. This information is computed from a software model driven by hardware counters. It is also possible to measure the GPU power consumption [Len+13]. Energy consumption of CPU and each different component can be also approximated through models [BM12], [TC+14]. However, not all hardware components (CPU, RAM, networks cards, switches, etc.) are equipped with built-in sensors. Alternatively, external power meters can monitor whole nodes (and also other equipment like switches or routers): for instance, [Ass+12] describes a deployment of power meters on three clusters; [Ros+14] proposes a software framework that integrates power meters in datacenters. While an external power meter should be more precise in measuring the total consumption (as the information measured is closer to what the electric utility measures through their electric meters), when a few jobs share a single node, it is not clear how each job should be accounted for the usage. Hackenberg et al. in [Dan14] introduce a high definition energy efficiency monitoring infrastructure that focuses on the correctness of power measurements and derived energy consumption calculations. This infrastructure is based upon temporal resolution optimizations through internal BMC polling and querying via IPMI. They demonstrate improved accuracy, scalability and low overhead with no usage of external wattmeters. They also describe the architecture of new FPGA based measurement infrastructure with spatial granularity down to CPU and DRAM, temporal granularity up to 1000 sample/s and accuracy target of 2% for both power and energy consumptions. SLURM has recently been enhanced by introducing the capability to regularly capture the instantaneous consumed power of nodes [Geo+c]. Based upon this power-aware framework, SLURM is the RJMS to provide energy accounting and power profiling per job. However, we argue that since power measurements take place only at the node level, the derived energy calculation will not reflect reality if jobs make use of nodes' parts or if they share nodes with other jobs.

6.1.2 Saving energy in HPC

Currently, there are two main approaches to improve energy efficiency in HPC: static power management, or designing hardware operating on efficient energy levels (e.g. low voltage CPUs used in IBM BlueGenes); and dynamic power management in which software dynamically switches the voltage and frequency (DVFS) used by a component [Gio13]. Designing efficient hardware is orthogonal to the scope of this chapter as we assume that an HPC platform is given; thus below we review the dynamic approaches. The core idea is to lower the frequency of the processor (which lowers the power consumption) when the job enters a computationally-light phase (recent surveys are [Gio13; Mit14]). DVFS can be used by the cluster scheduler without knowing the workload of an application (the frequency is dynamically

adapted to the load on each processor); however, the energy savings of such black-box approaches are limited (a review [Gio13] reports energy savings on NAS Parallel Benchmark of up to 20-25% with roughly 3-5% performance degradation).

Other methods complement DVFS. A simple, system-level technique is to switch off unused processors—here the key problem is the energy needed to switch the processor back on [Alb10]. The savings increase when a whole node, or a whole rack is switched off.

Further savings require adapting the application. For instance, in a distributed application that cannot be perfectly load-balanced, processors assigned smaller loads can be slowed down (by DVFS) to finish in roughly the same time as more loaded processors [Kap+05]. In distributed algorithms, communication between nodes is expensive in terms of energy; communication-optimal [Dem+12] or communication-avoiding algorithms reduce communication (sometimes increasing the amount of per-core computation).

To summarize, apart from black-box DVFS (which results in limited gains) and switching off the idle resources (of limited use, since most modern supercomputers are constantly overloaded), the approaches require the programmer to instrument the code, or even to change the algorithms. Moreover, saving energy incurs performance loss. Thus, a user must have incentives for saving energy. EnergyFairShare gives higher priority, thus lower queuing times, for users with smaller energy needs.

6.1.3 Fairness in HPC resource management

Fairness, even restricted to HPC, is a vast research area. The most popular approach is the *max-min fairness* [Gho+11], in which the goal is to maximize the performance of the worst-off user. Production schedulers like LSF [KC03], Maui/Moab [Jac+01], TORQUE [1] or SLURM [Yoo+03], use this approach through the FairShare algorithm. Usually the scheduler accounts for each CPU-second used by each user, decayed over time. Users with small total CPU-second usage have priority over users with large usage. Fair-share is compatible with the typical workflow of a scheduler (assign priorities, sort jobs, schedule according to priorities); moreover, the priorities can be further modified by static site policies (e.g., weighting groups of users in function of their payments to the site). EnergyFairShare uses FairShare, but the users are accounted for joules (watt-seconds), instead of CPU-seconds.

Many alternatives to FairShare were proposed; here we just list a few recently proposed. Klusaček et al. [KR13] modify conservative backfilling to improve fairness. Emeras et al. [Eme+14] proposes an algorithm optimizing the slowdown (the stretch) of each user's workload. The algorithm uses the concept of a virtual schedule, in which CPUs are assigned to users' workloads; thus it can be modified to treat the energy as the primary resource in a similar way as we modify FairShare.

[1] `http://www.adaptivecomputing.com/products/open-source/torque/`

EnergyFairShare can manage heterogeneous resources, as each resource can be characterized by its energy needs. Klusaček et al. [KR14] reviews multi-resource fairness. Papers differ by their definition of what a multi-resource fair schedule is [JW+13], and more specifically on the properties that their algorithm guarantee. The Dominant Resource Fairness algorithm proposed by Ghodsi et al. [Gho+11] guarantees sharing incentive, strategy proofness, envy freeness and Pareto efficiency. The algorithm proposed by Klusaček et al. [KR14] guarantees multi-resource awareness, heterogeneity awareness, insensitivity to scheduler decisions, walltime normalization, support for multi-node jobs. TORQUE and Maui/Moab support multi-resource fairness by counting resource usage using a measure different from CPU-seconds to count resource usage. TORQUE computes distance to a standard (mean) job. Maui/Moab [Jac+01] computes Processor-Equivalents, transforming consumption of non-standard resources to normalized CPU-seconds.

To our best knowledge, there is no other work proposing fair resource sharing based on energy. In the following section we argue that a energy fairsharing implies a multi-resource fairsharing.

6.2 EnergyFairShare Algorithm

EFS modifies FairShare to consider energy instead of CPUs as a main resource. Therefore, we start by describing the environment in which EFS works (a resource management system); then we describe the classic FairShare algorithm; and finally—the principle and the implementation of our algorithm. To better ground our discussion, next to discussing ideas behind these, we will show how they are realized in SLURM (the resource manager in which we implemented EFS).

6.2.1 Scheduling in Resource and Job Management Systems

Scheduling in a standard RJMS (such as SLURM, Maui, etc.) consists of two successive phases. First, pending jobs are prioritized according to some criteria. Then, picking jobs one by one in order of the assigned priorities, the scheduler assigns resources to jobs. EnergyFairShare is a prioritization algorithm; thus existing, efficient algorithms (such as backfilling [MF01b]) may be used in the second phase.

The priorities computed in the first phase are usually a linear combination of factors based on various parameters of a job. Example factors include job's waiting time (the longer the job queues, the higher is its priority); job's size (priority of long jobs may be reduced to increase job throughput); job's owner (to prefer accounts associated with a project that funded the supercomputer). Fair-share, described in the next section, may be used as an additional factor. Usually, a weight is assigned to each of the above factors. Weights allow to enact a policy that blends a combination of any of the above factors in any desired portion. For example, a site could configure

FairShare to be the dominant factor while setting job size and age factors to each contribute a smaller part.

6.2.2 Computing Priorities by Fair-Share

The FairShare algorithm computes queued jobs' factor (priority) based on the amount of resources consumed by the job's owner in the past. The job's FairShare factor is commonly added to other factors described in the previous section. The FairShare factor serves to prioritize queued jobs such that those jobs charging accounts that are under-serviced are scheduled first, while jobs charging accounts that are over-serviced are scheduled when the machine would otherwise go idle. Also, FairShare generalizes to hierarchies of accounts so that not only the owner's usage, but her group's, or her supergroup's, is taken into account (here, for simplicity of presentation we will not discuss it; we assume that each user has a single account and that the scheduling policy is based on these accounts).

Basically there are two parameters that influence SLURM FairShare factor: i) the normalized shares as defined in the associations of the database; and ii) the normalized usage of computing resources as a continuously evolving parameter computed from the accounting database.

A scheduling policy defines a target share s_u of a system for each account u. The algorithm computes, for each account u, the normalized share S_u as

$$S_u = s_u/(\sum_v s_v). \tag{6.1}$$

S_u expresses the share of the system that, on the average, user u is entitled to use.

The usage is computed as a total amount of consumed resources normalized by the amount of available resources; usually, recent usage counts more than the past (the usage is decayed over time). To compute the normalized usage, once every job completes, a RJMS stores in an accounting database the job's runtime multiplied by the amount of CPUs assigned (CPUs, as historically this was the most contested resource). The raw usage R_u for user u is computed based on a half-life formula that favors the most recent usage data. Past usage data decreases based on a single decay factor, D:

$$R_u = R_u(\delta_0) + D \cdot R_u(\delta_1) + D^2 \cdot R_u(\delta_2) + \ldots, \tag{6.2}$$

where δ_i is the ith measurement period counting from the current time moment (e.g.: δ_0 is the last 24 hours; δ_1 is the previous 24 hours etc.); and $R_u(\delta_i)$ is the number of CPU-seconds used by u during period δ_i. To get the normalized usage U_u, R_u is normalized by the total amount of available resources decayed over time,

$$U_u = R_u/(\delta_0 \cdot m + D \cdot \delta_1 \cdot m + D^2 \cdot \delta_2 \cdot m \ldots), \tag{6.3}$$

where m is the number of available CPUs. For instance, assume that on a $m = 50$ CPU system only the last 100 hours are taken into account ($\delta_0 = 100 \cdot 3600$, $D = 0$). If, during this period, a user completed 5 jobs (30-hour long) each taking 10 CPUs, her normalized usage is $(5 \cdot 30 \cdot 10)/(50 \cdot 100)$, or 0.3.

Various functions are used to convert share and usage to a priority value. For instance, SLURM computes the FairShare factor F_u for a user u as:

$$F_u = 2^{-U_u/S_u/d}, \tag{6.4}$$

where d is an additional damping parameter. Consequently: a user with no usage gets FairShare factor of 1; a user with usage equal to her share gets the factor of 0.5; and a user whose usage vastly exceeds her share gets the factor close to 0.

6.2.3 EnergyFairShare: the principle

EnergyFairShare, the algorithm we propose, uses the FairShare algorithm described above, but counts Joules (energy over time, Watts per second) instead of CPU-seconds. Thus, the accounting module keeps track of the energy consumed by each job, which requires it to get the data from the cluster's energy monitoring system. Then, the job's EFS priority is computed according to the owner's energy usage and its assigned share of the total energy budget (just as in the FairShare algorithm). The resulting value may be treated just as the FairShare priority is, so it may be added to other factors (job age, size, or even classic, CPU FairShare), to get the final priority.

6.2.4 EnergyFairShare as a SLURM scheduling feature

We implemented EFS upon the open-source resource and job management system SLURM [Yoo+03].

SLURM is designed as a client-server distributed application: a centralized server daemon, also known as the controller, communicates with a client daemon running on each computing node. Users can request the controller for resources to execute interactive or batch applications, referred as jobs. The controller dispatches the jobs on the available resources, whether full nodes or partial nodes, according to a configurable set of rules.

The SLURM controller also has a modular architecture composed of plugins responsible for different actions and tasks such as: job prioritization, resources selection, task placement, or accounting. We modified two of these plugins: the accounting plugin, to gather energy usage data; and the job prioritization plugin, where EFS is implemented.

SLURM has a particular plugin dedicated to gather information about the usage of various resources per node during job execution. This plugin, which is called *jobacct_gather*, collects information such as memory or CPU utilization of all tasks on each node. Then, the values are aggregated across all the nodes on which a job is running; then, two values per job are returned: the maximum and the

Trace	Config	efficiency	Stretch normalized by baseline				
			Min	Mean	Med	Std Dev.	Max
SDSC	fs	both	1.00	1.00	1.00	0.00	1.00
SDSC	efs	green	0.27	0.91	0.92	0.33	1.82
SDSC	efs	gluttonous	0.27	1.10	1.02	0.31	1.66
SDSC	fs+efs	green	0.27	0.89	0.95	0.24	1.17
SDSC	fs+efs	gluttonous	0.27	1.03	1.01	0.34	1.92
PIK	fs	both	1.00	1.00	1.00	0.00	1.00
PIK	efs	green	0.73	0.97	1.00	0.07	1.00
PIK	efs	gluttonous	0.97	1.03	1.00	0.11	1.47
PIK	fs+efs	green	0.89	1.00	1.00	0.03	1.00
PIK	fs+efs	gluttonous	1.00	1.03	1.00	0.10	1.45
Curie	fs	both	1.00	1.00	1.00	0.00	1.00
Curie	efs	green	0.14	1.08	1.00	0.64	3.44
Curie	efs	gluttonous	0.33	3.02	1.02	6.25	25.7
Curie	fs+efs	green	0.14	1.63	1.00	2.45	13.5
Curie	fs+efs	gluttonous	0.17	1.57	1.00	2.06	11.2

Table 6.1. Results of the simulations. In each simulation, one of the 20 most active users is either 30% more energy-efficient (green) or 30% less energy-efficient (gluttonous) than the rest. We compute the stretch for each job, normalized by the stretch in the baseline scenario; then average it over all jobs belonging to the same user. Rows presents statistics over 20 users.

average. These values can be then used for accounting, monitoring or scheduling. We extended this plugin to collect information from energy consumption sensors (however, we recall that measuring energy has its limitations, see Section 6.1.1).

Various plugins can be used for job prioritization; we implemented EFS as an extension to the *multifactor* plugin, since the plugin uses the prioritization framework described in Section 6.2.1 and since it implements the FairShare algorithm. For a job, the result of EFS is treated as a factor, and added to other factors for a job, such as age or size.

6.3 Experiments

We performed three different kinds of experiments with EFS. First, we implemented EFS in a simulator to run various traces and to check how changing jobs' energy efficiency influences their stretch. Second, we tested the EFS implemented as a SLURM extension: we emulated a particular supercomputer (Curie, a 80640-core machine), and added to its trace users with varying (simulated) energy efficiency. Third, we tested the whole approach—from collecting energy usage to making scheduling decisions—by running the EFS-SLURM extension on a real, albeit small-scale, cluster.

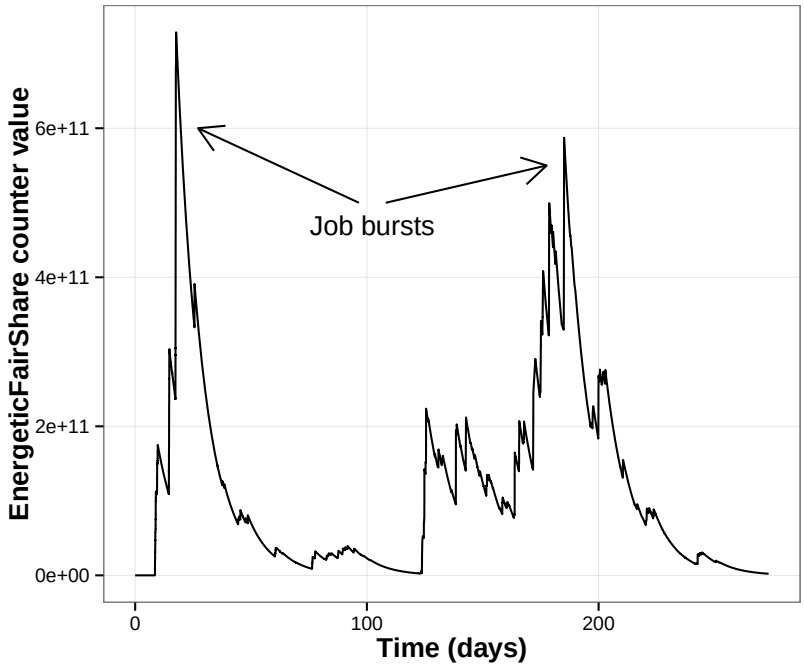


Figure 6.1. Evolution over time of EnergyFairShare counter for user 33 during the experiment simulating Curie with the EFS policy. This user, even being 30% more energy-efficient than other users, worsens her mean stretch by 344%.

6.3.1 Algorithm validation through simulations

We implemented EFS in Pyss [Pys], a discrete event simulator of batch schedulers. Once jobs are prioritized, the simulator uses the EASY backfilling to allocate resources. Waiting jobs are prioritized by three different policies:

- FairShare (FS): jobs owned by the user with the smallest recent CPU-second usage are prioritized (Section 6.2.2). We set equal target shares S_u and the decay factor D to one week, as it is the default value for SLURM.
- EnergyFairShare (EFS): jobs owned by the user with the smallest recent energy (Watt-seconds) usage are prioritized (Section 6.2.4). We set equal target shares and the same decay factor as FS.
- FS+EFS: FairShare and EnergyFairShare are normalized to their maximum current values and then summed.

All algorithms use job's arrival order to break ties.

We simulated three traces from the Parallel Workloads Archive [2]. We selected traces with high average usage to stress the scheduler (as in a lightly-loaded system, almost all jobs can be started immediately). Traces span different scales of HPC systems. The Curie trace is a 6 months trace of a 80640-cores machine ranked 26th on the top500 list of June 2014. The PIK trace is a 40 months trace of a 2560-cores cluster. Finally, the SDSC SP2 trace is a 24 months trace of a 128-cores cluster. We use the cleaned version of each trace. Results are presented in Table 6.1.

[2] `http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`

As it is possible to measure the nodes' energy consumption only relatively recently, the traces do not have the information about the energy consumption of jobs. In each experiment, we set the energy consumption to be proportional to the job's CPU-seconds. To get the baseline result, we first simulate each trace with each policy, where all job have the same energy efficiency. Then, we select the 20 users that submit the biggest number of jobs. For each selected user, we improve their jobs' energy efficiency by 30% (thus, a conservative estimate, since black-box DVFS report a 20-25% gain, Section 6.1.2) and run the whole simulation (keeping other users not modified). Then, for each job of this user, we compare its stretch with the stretch in the baseline result by computing the factor `stretch-green / stretch-baseline`. Finally, to get an influence of the policy on the user, we compute the average of these factors (for all users' jobs). In Table 6.1, the "green" rows show statistics (the minimum, the average, the median, the standard deviation and the maximum) of average factor on the sample of 20 users. Similarly, the "gluttonous" rows show statistics when the efficiency is *worsened* by 30% for each job.

As we expected, EFS (slightly) improves stretches of energy-efficient users. The difference is most visible on the SDSC trace, in which EFS reduces the mean stretch by 9% for the green users; and increases by 10% for the inefficient users. On the PIK trace, EFS reduces the mean stretch by 3% for the green users; and increases by 3% for the inefficient users. However, in the Curie trace, the results are less clear: although EFS distinguishes between green and gluttonous users (gluttonous users have, on the average, their stretch increased 3 times), green users are apparently 8% worse-off than in the baseline.

The worst-off "green" user has her stretch increased by 344%. We studied this user in detail. Figure 6.1 presents the evolution of the internal EnergyFairShare usage counter (U_u) through time. We see two peaks in the usage, corresponding to huge amounts of energy consumed by the users' jobs (we annotate these moments as job bursts). This implies that even with better energy efficiency, the user will have a high energy penalty at this moment of the trace. To be launched by the system, her jobs will have to wait until the usage decreases. We observed similar effects for other users.

Another effect that influences the Curie results is that the trace is very volatile—the standard deviations are, approximately, an order of magnitude greater than in SDSC and PIK. In this large system a small change in the scheduling decision can lead to a totally different schedule. For example, let us imagine a queue with a small and a huge job (the trace has quite a few jobs using 64.000 cores for more than an hour). If the scheduler chooses the huge job to be launched first, many other jobs will be delayed. Whereas if the small one is choosen first, thanks to backfilling, more jobs will be able to run before the huge job.

6.3.2 A Real implementation on an emulated platform

In this subsection, we test the implementation of our algorithm as a SLURM extension by emulating a large-scale supercomputer, Curie (a 5040 nodes, 80640 cores machine). Our experimental platform enables us to emulate 5040 nodes on 20 physical nodes by running multiple `slurmd` daemons (each deamon corresponds to a single emulated node). We run an unmodified SLURM to manage the emulated nodes, but jobs are emulated using `sleep` commands. Thus, we had to modify the way SLURM collects energy measurements: as the platform is emulated, the energy consumed by each job is not measured through sensors, but instead we inject it directly based on the trace.

We use Light-ESP workload [GH12]. Light-ESP is based on ESP [Won+00], a synthetic benchmark workload consisting of 230 jobs of 14 types. Light-ESP reduces the runtime of these jobs, so that the turnaround time of the whole log is 15 minutes. We repeated Light-ESP workload 4 times.

We compared the job stretches under four prioritization policies: FairShare (FS), EnergyFairShare (EFS), FS+EFS; and FIFO, in which jobs' priorities are based on their waiting time. As in the previous experiment, and as in a default SLURM configuration, an EASY backfilling algorithm is used to allocate processors to jobs.

The Light-ESP workload does not specify jobs' owners and jobs' energy consumption, thus we will study two cases. First, we add three users having the same workload, but varying energy efficiency. We expect that the energy-efficient user should have a smaller stretch than the inefficient one. Second, we add three users having different workload and the same energy efficiency. We expect that EFS will work in the same way as FS.

3 users with the same workload and different energy efficiency

In this experiment, we add to the Light-ESP trace three users with the same workload but with different energy efficiency. User 1 is a user who optimized by 30% the energy efficiency of her job, user 2 is a normal user, and user 3 is inefficient (by 30%). Figure 6.2 presents results. In FS and FIFO policies, all users have roughly the same stretch which is expected as all users have the same workload. EFS and FS+EFS reward efficient users and punish the inefficient ones. User 3, who consumed the most energy, has also the highest mean stretch; whereas the efficient User 1 has a stretch reduced by roughly 60%. Compared to EFS, FS+EFS policy has smaller difference between each user's mean stretch because FS+ESF is a combination of the two above policies.

From a global point of view, policies perform equally well, as they all achieve a utilization of about 89%.

3 users with different workload and the same energy efficiency

In this experiment, we add to the Light-ESP three users with different workload (and the same energy efficiency). Here, we want to show that our algorithm is equivalent to the classic FairShare, if jobs have the same energy efficiency. On

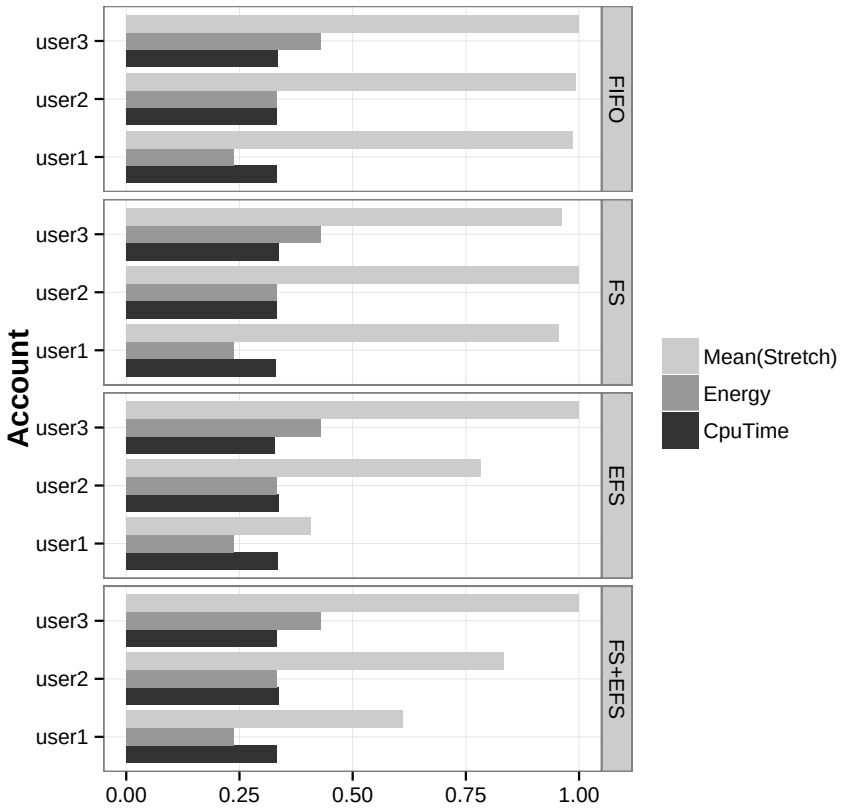


Figure 6.2. SLURM implementation. Emulation of Curie cluster running 4 Light-ESP traces. Three users have the same workloads, but different energy efficiency. All values shown are normalized.

Figure 6.3, we can see that, except for FIFO, each policy has the same results. User 3, the user with the highest workload (and also—the highest energy consumption) is penalized compared to other users. These variation of stretch for each user time does not change the global performance of the algorithm as each policy have the same mean utilization of 89%.

6.3.3 Experiments on a real platform

In this subsection we perform experiments on a real, small-scale cluster using monitored energy consumption data. Our objective is to evaluate the new EFS scheduling strategy as implemented upon SLURM and validate its effectiveness under real-life conditions. The experiments have been performed upon a small BULL cluster dedicated for R&D experiments. The cluster is composed of 16 nodes with Intel Xeon CPU E5649 (2 sockets/node and 6 cores/socket), 24GB of memory and Infiniband network. For the sake of the experiments, we installed our modified SLURM version, configured one dedicated SLURM controller, 15 compute nodes and activated the energy accounting framework with monitoring through IPMI method.

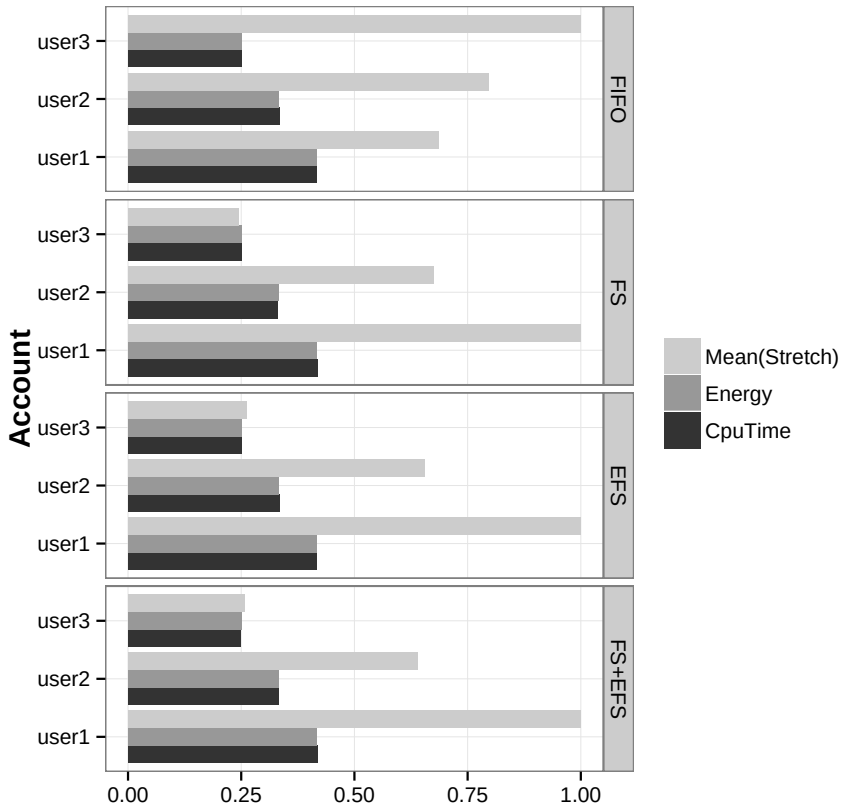


Figure 6.3. SLURM implementation. Emulation of Curie cluster running 4 Light-ESP traces. Three users have different workloads, but the same energy efficiency. All values shown are normalized.

Our experiments follow the same guidelines as the emulation-based experiments in the previous subsection 6.3.2 using the 4x Light-ESP benchmark as an input workload. Table II presents the characteristics of a single Light-ESP workload as adapted for the 180 cores of our cluster. There are two important differences compared to the workload used in 6.3.2: i) The energy consumption, within SLURM, is measured at the level of nodes (as described in Section 6.2.4), which means that only exclusive allocations can be considered (i.e., two jobs cannot share a single node). Thus, the size of each job type in Table II is a multiple of 12 cores. ii) In order to observe actual energy consumption, instead of `sleep` commands, we have to execute real jobs. We used Linpack MPI applications that we calibrated to fit the target run-time of the benchmark.

Before starting the experiments we have also profiled Linpack for the different job classes in terms of energy-performance tradeoffs. Figure 6.4 provides graphs of the evolution of these tradeoffs when changing CPU frequencies of processors. We show just a representative part of table's II job types' observed behavior. It is interesting to see that the lowest frequency is not the most energy-efficient. On

	Job Size (cores)	Number of Jobs	Run Time (sec)
Job Type	Job size for cluster of 180 cores (Fraction of job size relative to system size)/ Number of Jobs / Run Time (sec)		
A	12 (0.03125)	75	22s
B	12 (0.06250)	9	27s
C	96 (0.50000)	3	45s
D	48 (0.25000)	3	51s
E	96 (0.50000)	3	26s
F	24 (0.06250)	9	154s
G	36 (0.12500)	6	111s
H	36 (0.15820)	6	89s
I	12 (0.03125)	24	119s
J	24 (0.06250)	24	60s
K	24 (0.09570)	15	41s
L	36 (0.12500)	36	30s
M	48 (0.25000)	15	15s
Z	180 (1.0)	2	20s
Total Jobs / Theoretic Run Time	230 / 935s		

Table 6.2. Synthetic workload characteristics of Light-ESP benchmark [GH12] as adapted for a 180 cores cluster with exclusive nodes allocations.

the contrary: the most energy efficient CPU frequency varies between 2.3 Ghz and 2.2 Ghz for most cases; whereas the lowest CPU frequency (1.6 Ghz) is usually the most energy-consuming one. Note that this behavior might be specific to the type of the job used—a heavily-optimized, computationally-intensive Linpack. Also take into account that we are here effectively using a black-box approach, as we are scaling the whole application.

We divided the jobs from Light-ESP into three groups of equal sizes; then, we assumed that each group corresponds to a single user having different energy-efficiency.

The first user is “green”—she takes into account the tradeoffs analysis and adapts her CPU frequency to the optimal value per each job type. The second user is “normal”—he selects the highest frequency in order to optimize performance. Finally, the third user is “gluttonous”—he chooses the slowest CPU frequency resulting in the worst energy efficiency.

Each job is submitted in a particular moment in the workload. To be certain that the order of job submissions does not influences the results, we are launching 4 times the same workload (light-ESP); each time changing the attributed user per group of jobs.

Figure 6.5a shows the cumulative distribution function (CDF) of stretch with only EFS activated. “Green” user’s jobs have smaller stretch than normal user. However, it appears that the “green” and the “gluttonous” users have quite similar stretches. Even if this may seem surprising at first it is explained by the fact that in our context the gluttonous jobs are represented by the lowest CPU-Frequency. This results in high energy consumption because of very large run time. Hence even if

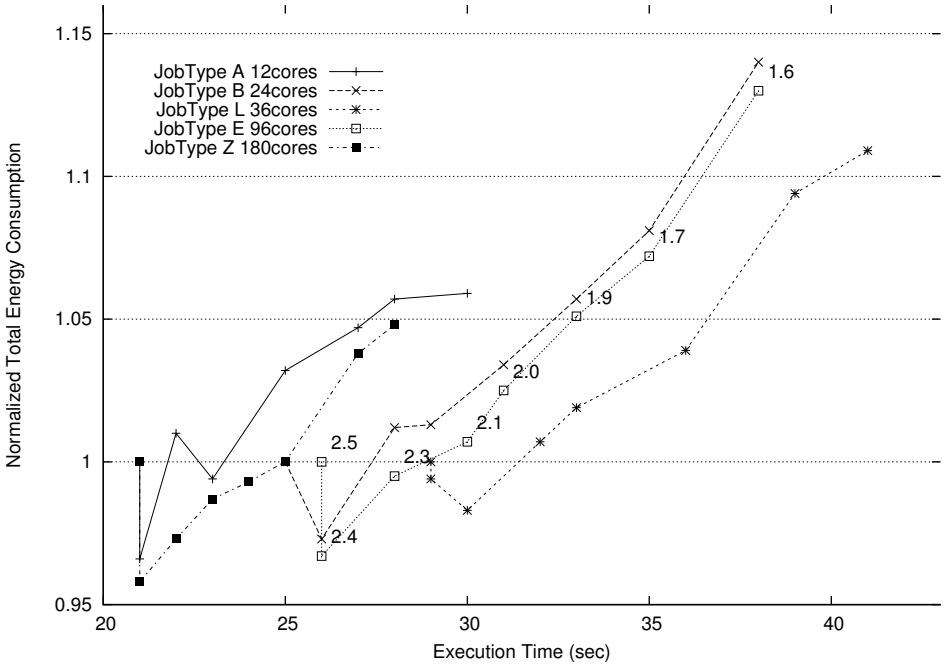


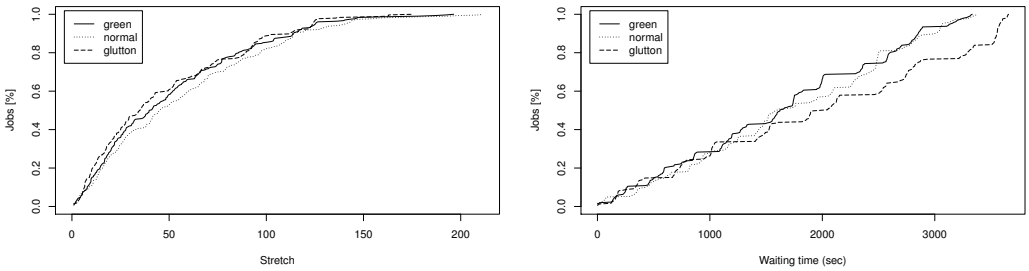
Figure 6.4. Performance vs. energy tradeoffs for Linpack applications as calibrated for Light-ESP job types (table II) running on a 180-cores cluster at different frequencies.

gluttonous jobs have larger queue waiting times than green jobs because of EFS, they have also larger running times, which results in lower stretches.

Figure 6.5b shows the CDF of jobs' waiting time (the time a job spends in the queue) in the case of EFS policy. We can see that jobs are executed in groups according to the users' EFS factor.

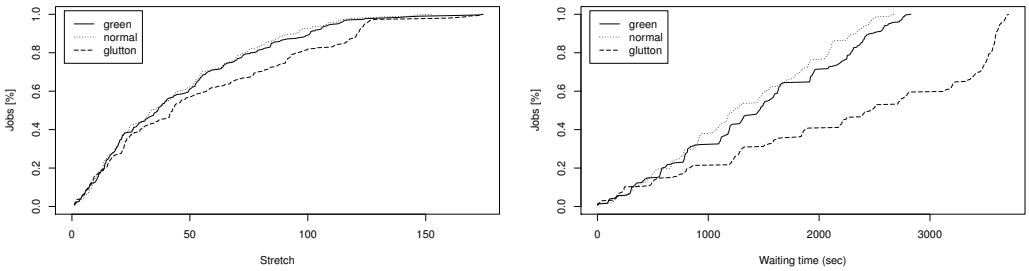
Jobs whom waiting time is up to approximately 1250 seconds have similar waiting time regardless of the energy efficiency. The influence of the efficiency is visible on longer-waiting jobs. When jobs are continously submitted, the EFS policy translates to a scheduling policy that executes a few jobs of one user; then a few jobs of another user, etc. When a user becomes the one with the minimal EFS factor, the remaining user's jobs are started as long as the user's EFS factor remains lower than the other two users. Eventually, the energy consumed by the newly started jobs adds up, and another user becomes the one having the minimal factor. The number of executed jobs per such "turn" depends on the consumed energy. That is why the gluttonous user eventually is waiting much longer than the other two users. The green users' jobs have slightly smaller waiting time than the normal users' jobs.

Figure 6.6 shows the CDF of stretch and waiting times with the same workload and jobs execution as previously, but using the original FairShare (FS). In this case we consider only CPU-time for prioritization. Both in terms of stretch and waiting times we can see that "normal" user's jobs have the best results since their jobs have the optimal performance. "Green" user's jobs have quite good results whereas "gluttonous" user's jobs have the worst result because of large run times.



(a) CDF on Stretch with EFS **(b)** CDF on Waiting time with EFS

Figure 6.5. Cumulated Distribution Function for Stretch and Waiting time with SLURM EnergyFairShare policy running LightESP x4 workload with Linpack executions by 3 users with different energy efficiencies.

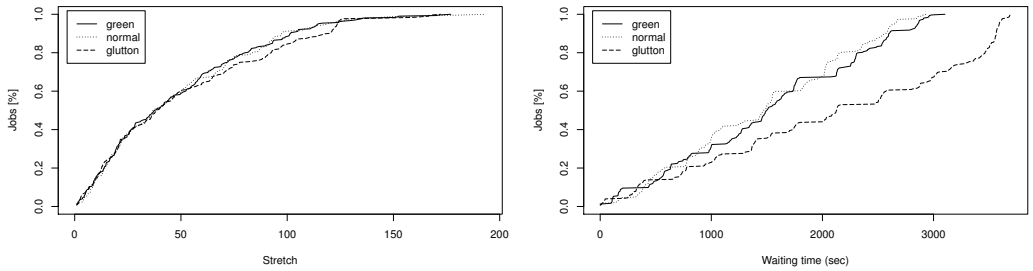


(a) CDF on Stretch with FS **(b)** CDF on Waiting time with FS

Figure 6.6. Cumulated Distribution Function for Stretch and Waiting time with SLURM FairShare policy running four LightESP workloads with Linpack executions by 3 users with different energy efficiencies.

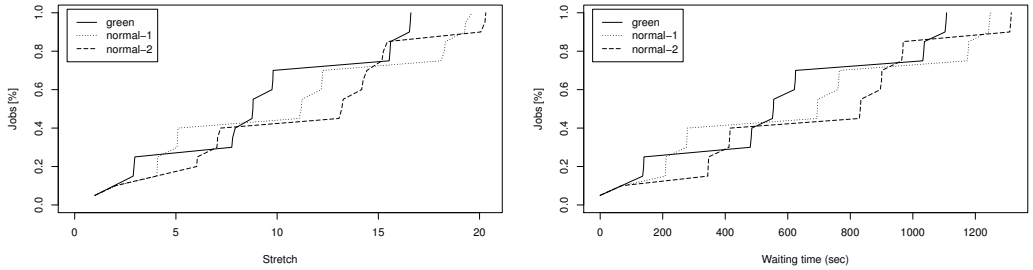
In the following experiment we have activated both FS along with EFS with equal weights within SLURM. This means that both CPU-time and energy consumption can play equal role in the calculation of job's priority. Figure 6.7 shows the CDF of stretch and waiting times with light-ESP x4 workload with FS+EFS policies. "Gluttonous" user's jobs have an important disadvantage when compared to the other two groups, especially in terms of waiting times. In contrast with 6.5a, figure 6.7a shows a significant degradation for "gluttonous" user's jobs and this is because the waiting times are much longer in the FS+EFS case as we can see in 6.7b. Furthermore, we can observe that "green" and "normal" user's jobs provide similarly good results. When "green" jobs gain in energy, "normal" jobs gain equally in CPU-time. However, we can observe a slight advantage of "normal" user's jobs. This can be explained by the fact that the difference in gained CPU-time of "normal" jobs are more noticeable than the gains in energy consumption by the "green" jobs.

Finally we performed another experiment with a simple workload of 60 jobs executing the same Linpack application upon 60 cores each. The jobs were separated into 3 groups where one group is launched as energy efficient with the optimal



(a) CDF on Stretch with FS + EFS **(b)** CDF on Waititng time with FS + EFS

Figure 6.7. Cumulated Distribution Function for Stretch and Waiting time with SLURM FairShare plus EnergyFairShare policies running four LightESP workloads with Linpack executions by 3 users with different energy efficiencies.



(a) CDF on Stretch with EFS **(b)** CDF on Waiting time with EFS

Figure 6.8. Cumulated Distribution Function for Stretch and Waiting time with SLURM EnergyFairShare policy running a submission burst of 60 similar jobs with Linpack executions by 1 energy-efficient and 2 normal users

CPU-Frequency (2.3GHz) and the other two have been launched both with normal characteristics (one with larger CPU-Frequency 2.5GHz and one using on-demand governor). All the jobs have been launched simultaneously on the cluster. The difference with previous figures (in addition to having of 2 normal users), was that we have calibrated Linpack to be executed with good energy-efficiency when running with DVFS of 2.3 GHz. The results are presented in form of CDF on stretch and waiting times in figures 6.8a and 6.8b. The figures show a significant improvement in both stretch and waiting times for the green user's jobs. Hence, in these figures, the higher energy-efficiency of the green jobs is rewarded with optimized stretch and waiting times.

6.4 Conclusion

In this chapter, we proposed EnergyFairShare, an algorithm to prioritize jobs based on their owners' past energy usage. The main goal of our policy is to explicitly manage what is currently an important cost of running an HPC resource: the electricity. In general, the more electricity a user consumed in the past, the lower

the priority of her future jobs is. We implemented the algorithm in a simulator, and as a plugin for SLURM, a popular HPC resource manager. Our implementation of scheduling policies has been released in SLURM version 15.08. We verified experimentally that more energy-efficient jobs have lower stretches. We claim that EnergyFairShare should motivate users to make their jobs more energy-efficient, in the same way as FairShare incentives users to make their jobs more CPU-runtime efficient—however, to test this claim, our mechanism would need to be used in a production system.

Utilization of any resource implies energy consumption, thus a possible objection to our work might be that we penalize "large" jobs that should be executed anyway (after all, the interest in doing HPC are large-scale calculations). However, we use the same principle as the one already used for managing CPU-seconds—FairShare. Scheduling policies employing FairShare are very common in existing HPC centers; indeed, FairShare policies penalize users submitting large jobs; but if a particular user demonstrates that the size of her job is a consequence of a true, scientific need (and not — inefficiency), the administrators can increase her target share. Same argument applies to EnergyFairShare.

Not all resources consume the same amount of energy, thus some resources may be more "costly" to users in terms of EFS. We claim that this heterogeneity should push users to run jobs on resources that are efficient for these jobs.

Of course, a scheduler-level prioritization mechanism is not sufficient to make users save energy. First, users need to be aware of the total energy consumption of each of her jobs; and, perhaps additionally, about their average energy efficiency (number of Watts consumed during an average CPU-second), compared to other jobs in the system. EFS already stores the information needed to compute these values; they can be presented in addition to existing accounting. Second, energy profilers should help to tune jobs for energy-efficiency (just as standard profilers help to tune for the run-time).

Finally, to be energy-efficient, we must first precisely measure the consumed energy. Currently, it is possible to measure either the whole node (which is a problem when a node is shared by a few jobs), or some components (CPU, GPU)—but not the others (network, memory, storage). However, as energy efficiency is a key to performance and to low running costs, we envision that more and more precise measures will be available.

Acknowledgment

We thank the providers of the workload logs we used in simulation: Joseph Emeras (Curie), Ciaron Linstead (PIK IPLEX) and Victor Hazlewood (SDSC SP2). The work is partially supported by the ANR project called MOEBUS and by the Polish National Science Center grant Sonata (UMO-2012/07/D/ST6/02440).

7 Improving Backfilling by using Machine Learning to Predict Running Times

> *Tell me and I forget. Teach me and I remember. Involve me and I learn.*
>
> **— Benjamin Franklin**

More and more data are produced in HPC systems by monitoring the platform (CPU usage, I/O traffic, energy consumption, *etc.*), by the job management system (*i.e.*, the characteristics of the *jobs* to be executed and those which have already been executed) and by analytics at the application level (parameters, results and temporary results). All this data is most of the time ignored by the actual systems for scheduling jobs.

The technologies and methods studied in the field of *big data* (including Machine Learning) could and must be used for scheduling jobs in the new HPC platforms.

For instance, and this will be the focus of this chapter, the running time of a given job on a specific HPC platform is usually not known in advance and moreover, it depends on the context (characteristics of the other jobs, global load, *etc.*). In practice, most job management systems ask the users for an upper bound on the job running time, threatening to kill it if it exceeds this requested value. This leads to very bad estimates of the running times given by the users [Tsa+05]. A precise knowledge of the running times is even more important since the algorithms used in most of these systems assume that this value is perfectly known. Thus, it is crucial to determine how to estimate the running times in order to improve scheduling. We believe that there is a huge potential gain in studying this question more deeply and provide more efficient scheduling mechanisms.

Obviously, the job running time is not the only parameter impacted by uncertainties. We focus on it as a proof of concept in order to show that it is possible to improve the scheduling performances for popular FCFS-BF (First Come First Serve with Backfilling) batch scheduling policy. The analysis provided on this work can be extended easily to other scheduling policies.

The main question addressed in this work is to determine to what extent predictions of the running times may help for obtaining a better schedule. For this purpose, we rely on on-line regression methods and consider a family of loss (or cost) functions that are used to learn the prediction model. Then, we show how to use the predictions obtained by improving popular scheduling algorithms. Finally, we perform an experimental evaluation of the proposed new algorithms using several

actual log datas on various platforms. The results show an average gain of 28% compared to the classical EASY policy (with a maximum of 86%) and 11% in average compared to EASY++.

The following chapter corresponds to a publication [Gau+] made in collaboration with Eric Gaussier, Valentin Reis, and Denis Trystram.

7.1 Problem description

7.1.1 Job management

We are interested in this work in scheduling jobs in HPC platforms. The application developers (or users) 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. In most HPC centers, these users are requested to provide some information about their applications in order to help the system to take better decisions. In particular, it is expected that they give an estimation of the running times. As a job is killed if its actual running time is greater than its requested running time, users tend to significantly increase the duration estimates [Tsa+05].

We focus on EASY backfilling as a basic mechanism for assessing our approach. In the remaining of the chapter, the corresponding policy will be denoted in short by EASY.

7.1.2 Dealing with uncertainties

The objective of this section is to show by some simulations on actual data that using good running time predictions significantly improves the scheduling performances. First, let us clarify the vocabulary: a job is *over-predicted* if its predicted running time is greater than the actual running time. Similarly, a job is under-predicted when the predicted running time is lower than the actual running time.

Table 7.1 reports the comparison of simulations based on the testbed detailed in Section 7.5. Each log runs with EASY, first with the original user's requested running times, then with the actual running times (as if the users were entirely clairvoyant). The metric used for comparison, AVEbsld is described in Subsection 7.4.3.

The results reported in Table 7.1 emphasize that clairvoyant simulations are in average 27% better than simulations using the original requested running times. As running times are shorter in the clairvoyant case, more jobs can be backfilled and thus, the performances are improved. Taking into account actual running time values always improves the scheduling performances, thus accurate running time estimates are crucial for reaching good performances.

7.1.3 Formulation of the problem

The problem studied in this work is to execute a set of concurrent parallel jobs with rigid resource requirements (it means that the number of resources required by

Log	EASY	EASY-CLAIRVOYANT
KTH-SP2	92.6	71.7 (22%)
CTC-SP2	49.6	37.2 (25%)
SDSC-SP2	87.9	70.5 (19%)
SDSC-BLUE	36.5	30.6 (16%)
Curie	202.1	69.9 (65%)
Metacentrum	97.6	81.7 (16%)

Table 7.1. AVEbsld performances of EASY (using requested times) and EASY-CLAIRVOYANT (using actual running times). Values between parentheses show the corresponding decrease in AVEbsld.

a job is known in advance) on a HPC platform represented by a pool of m identical resources (we do not assume any particular interconnection topology). The jobs are submitted over time (on-line).

There are n independent jobs (indexed by integers), where job j has the following characteristics:

- Submission date r_j (sometimes called *release date*);
- Resource requirement q_j (processor count);
- Actual running time p_j (sometimes called *processing time*);
- Requested running time $\widetilde{p_j}$, which is an upper bound on p_j;
- Additional data that has no direct impact on the physical description of the job (*e.g.* the time of the day when the job is submitted or the executable name). This data may be used to learn about jobs, users and the system.

The resource requirement q_j of a job is known when the job is submitted at time r_j, while the running time p_j is given as an estimate. Its actual value is only known *a posteriori* when the job really completes.

The problem is to design several algorithms that predict the running times in order to provide good scheduling performances.

7.2 Related Work

7.2.1 Prediction

Historically, job running time prediction has been first attempted [Gib97] by categorizing the jobs according to a predefined set of rules. Then, statistics based on such job's category are used to generate a prediction. In this approach, called *Templates*, a partitioning into templates has to be provided by the job management system or the system administrator. It can be seen as an ancestor of tree-based regression models in which the binning has to be obtained trough statistical analysis of the specific system and population and/or discussion with a domain expert. The technique was subsequently adapted [Smi+04] using a more automatic way (a

genetic algorithm evolving template attributes) to generate the rules. These works used minimal, high-level information about jobs similar to what can be found in HPC logs.

There exist other works that use more specialized methods, but require the modeling of the jobs. For instance, Schopf *et al.* predict running time of applications by analyzing their functional structure [Sch+99]. An another example is the method developed by Mendes *et al.* which performs static analysis of the applications [MR98].

A later survey [MF10] evaluates the use of more recent supervised learning tools. This work focuses on two scientific applications and uses in-depth information about both the jobs (*e.g.* input parameters) and the machines (*e.g.* disk speed). A closely related paper by Duan *et al.* [Dua+09] proposes an hybrid Bayesian-neural network approach to dynamically model and predict the running time of scientific applications. It uses in-depth information about jobs and their environment as well.

All these previous approaches assume jobs and their running times to be identically distributed and independent (*i.i.d.*), and therefore, they do not leverage dependencies between job submissions. A stochastic model [Nis06] has been proposed for predicting job running time distributions. By opposition to the previous studies which only used job descriptions, this technique only relies on historical running time information. It treats successive running times of a given user as the observations of a Hidden Markov Model [Rab89], and hence it does not use the hypothesis that job submissions would be *i.i.d.*.

7.2.2 Scheduling based on Predictions

There are only a few methods for performing scheduling using predictive techniques.

The Hidden Markov Model used in [Nis06] is a probabilistic generative model, and as such it is possible to easily obtain the conditional distribution of the running time of a job. As a consequence, it is possible to design scheduling algorithms that use job running time distributions as an input [NF08].

The most relevant work for scheduling jobs on large parallel systems using predictions is [Tsa+07], in which the average of the two last available running times of the job's user is used as a prediction. It leads to surprisingly good results given its simplicity. This work also introduces an improved version of EASY: EASY++. This algorithm use the predicted value for the backfilling. The waiting jobs are considered by their order of arrival, but during the backfilling phase jobs are sorted shortest first. They also introduce a correction mechanism: when the the prediction technique under-estimates a running time, a new estimation of the running time has to be obtained. The proposed correction mechanism is simple: the authors add a fixed amount of time from a predefined list of values each time they under-estimate.

To the best of our knowledge, there is no other work that relies on state-of-the-art machine learning methodology for running time prediction and evaluates the resulting scheduling. In the following two sections, we present prediction and scheduling mechanisms.

7.3 Prediction method

We first outline in this section the characteristics that the prediction method should have, prior to describing our approach in detail.

7.3.1 Rationale

Let us first argue that an approach based on machine learning for predicting the running times of jobs should have the following characteristics:

It should be based on minimal information. In hope that results extend well to new HPC systems and be usable in mainstream job management system, a learning-based system should prove its effectiveness on minimal job descriptions. In this light, one reasonable approach is to use information of the Standard Workload Format (*SWF*) [Fei+14].

It should work on-line. This is motivated by previous studies which emphasize that dependencies between job running times are so far from being *i.i.d.* that *two successive running times* [Tsa+07] are enough to predict running time with good accuracy. Therefore, an algorithm based on batches (which re-approximates the learned concept once every k jobs, where $k \gg 1$) should not be favored.

It should use both job description and system history Unlike previous studies that either assume jobs to be temporally independent or rely exclusively on running time locality [Nis06], a holistic approach should leverage both job descriptions and temporal dependencies.

It should be robust to noise. Because HPC jobs can have an erratic behavior (*e.g.* they may fail or hang-up), the data one is relying on will be noisy. The learning algorithm should be robust to this noise and avoid overfitting.

It should accept an arbitrary loss function. Attempting to minimize the cumulative loss of an arbitrary function allows for a "declarative" statement of the harm incurred by a misprediction. This last aspect is motivated by the intuition that an inaccurate prediction of a job's running time would not harm the scheduling in an identical way depending on the job's characteristics and the direction of the error. This will be explored in detail in the next Subsection. Arbitrary loss functions generally pose computational limitations, and in practice convex loss functions are often used.

We now turn to the description of the prediction method.

7.3.2 A new prediction approach

A job j is represented by a vector[1] $\mathbf{x}_j \in \mathbb{R}^n$ where n is the number of features of the model. The features which we feed the algorithm with are described in Table 7.2. These features are taken from various sources of information, such as the job's description ($\widetilde{p_j}$ and q_j). Others are taken from historical information (*e.g.* $p_{j-1}^{(k)}$) and some are taken from the current state of the system (*e.g.* Jobs Currently Running). Additionally, some features are taken from the environment (*e.g.* Time of the day).

The prediction is achieved via a ℓ_2-regularized polynomial model. This choice is motivated by the availability of highly robust algorithms to fit on-line linear regression models [Ros+13], even in the presence of an adversary scaling of the features. The ℓ_2 norm regularizer is used to prevent the quadratic model from overfitting and the polynomial representation (here of degree 2) to take into account dependencies between features. The regression function is:

$$f(\mathbf{w}, \mathbf{x}) = \mathbf{w}^\intercal \Phi(\mathbf{x}) \quad \mathbf{w} \in \mathbb{R}^{1+2n+\binom{n}{2}} \tag{7.1}$$

where the w_i are the parameters (to be learned) of the model, and Φ is a vector of basis functions:

$$\Phi(\mathbf{x}) = (1, x_1, \cdots, x_n, x_1x_2, \cdots, x_kx_l, \cdots, x_{n-1}x_n)^\intercal$$

Denoting the actual running time of job j by p_j, the cumulative loss for up to the N-th already-processed job is[2]:

$$\sum_{j=1}^{N} \mathfrak{L}(\mathbf{x}_j, f(\mathbf{x}_j), p_j)$$

where $\mathfrak{L}(\mathbf{x}, f(\mathbf{x}), y)$ is the loss function associated with predicting a value of $f(\mathbf{x})$ in the case where the actual running time is y. The regression problem finally takes the form:

$$\arg\min_{\mathbf{w}} \sum_{j=1}^{N} \mathfrak{L}(\mathbf{x}_j, \mathbf{w}^\intercal \Phi(\mathbf{x}_j), p_j) + \lambda ||\mathbf{w}||_2 \tag{7.2}$$

where λ is the regularization parameter. Once $\mathbf{w}$ has been learned, new running times are predicted through Equation (7.1).

The choice of the loss function is crucial and not straightforward here as the impact of a bad prediction on the running time varies from job to job as well as from the direction of the error (over- or under-prediction). Indeed, scheduling algorithms behave differently with respect to under-prediction and over-prediction: in the case of an under-prediction, a destructive effect on the planned schedule

[1] As common in machine learning, we use bold letters to denote vectors and standard letters for scalars.

[2] Note that one can also consider the k latest jobs or weigh differently the jobs to favor recent ones. These variants are straightforward to consider from the framework developed here.

Feature	Meaning
$\widetilde{p_j}$	the time the user requested for her job.
$p_{j-1}^{(k)}$	the running time of the last job of the same user, or 0 if such a job does not exist.
$p_{j-2}^{(k)}$	the running time of the second-to-last job of the same user, or 0 if N/A.
$p_{j-3}^{(k)}$	the running time of the third-to-last job of the same user, or 0 if N/A.
$AVE_2^{(k)}(p)$	the average running time of the two last historically recorded jobs of the same user.
$AVE_3^{(k)}(p)$	the average running time of the three last historically recorded jobs of the same user.
$AVE_{all}^{(k)}(p)$	the average running time of all historically recorded jobs of the same user.
q_j	amount of (CPU) resource requested by job j.
$AVE_{hist,r_j}^{(k)}(q)$	average historical resource request of user k, taken at release date of job j.
$\frac{q_j}{AVE_{hist,r_j}^{(k)}(q)}$	amount of resource requested normalized by average resource request.
$AVE_{curr,r_j}^{(k)}(q)$	average resource request of the user's currently running jobs, at release date
JOBS CURRENTLY RUNNING	number of jobs of the user running, at release date
LONGEST CURRENT RUNNING TIME	longest running time (so-far) of the user's currently running jobs, at release date
SUM CURRENT RUNNING TIMES	sum of the running times (so-far) of the user's currently running jobs, at release date
OCCUPIED RESOURCES	total size of resources currently being allocated to the same user.
BREAK TIME	time elapsed since last job completion from the same user.
$\begin{cases} cos(\frac{2*\pi}{t_{day}} * (r_j \mod t_{day})) \\ sin(\frac{2*\pi}{t_{day}} * (r_j \mod t_{day})) \end{cases}$	time of the day the job was released. The periodic feature is decomposed into its cosinus and sinus, using the day period t_{day} (length of a day in seconds)
$\begin{cases} cos(\frac{2*\pi}{t_{week}} * (r_j \mod t_{week})) \\ sin(\frac{2*\pi}{t_{week}} * (r_j \mod t_{week})) \end{cases}$	time of the week the job was released. The periodic feature is decomposed into its cosinus and sinus, using the week period t_{week} (length of a day in seconds)

Table 7.2. Features extracted from the *SWF* data, for job j, belonging to user k.

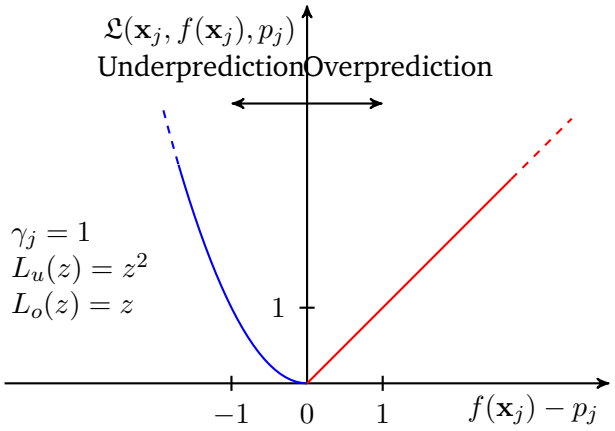


Figure 7.1. Example Loss function $\mathfrak{L}$, plotted with respect to the difference of it's second and third parameters $f(\mathbf{x}_j) - p_j$ (the prediction error).

can happen, while an over-prediction never makes a planned schedule feasible but may imply unused resources. This suggests that one should rely on asymmetrical losses, that can be based on standard loss functions dedicated to either under- or over-prediction. Another source of complication with respect to prediction arises not just from the backfilling strategy, but from the scheduling problem itself. The tasks have a two-dimensional representation based on (q_j, p_j) and it is reasonable to expect that the difficulty of finding a good schedule should be dependent not only on the prediction error, but also on how it is distributed among jobs of different q and p. This suggests that the loss function should be weighted differently according to the jobs considered, leading to:

$$\mathfrak{L}(\mathbf{x}_j, f(\mathbf{x}_j), p_j) = \begin{cases} \gamma_j . L_u(f(\mathbf{x}_j) - p_j) & \text{if } f(\mathbf{x}_j) \geq p_j \\ \gamma_j . L_o(p_j - f(\mathbf{x}_j)) & \text{if } f(\mathbf{x}_j) < p_j \end{cases}$$

where L_u is the underprediction basis loss function, L_o is the overprediction basis loss function, and $\gamma_j > 0$ corresponds to the weight of job j.

Figure 7.1 shows an example of an asymmetrical loss function with a unit weight γ_j, using a squared loss basis for underprediction and a linear loss basis for overprediction.

In this study, we consider two standard loss functions for under- and over-prediction, namely the squared loss ($L(z) = z^2$) and the linear loss ($L(z) = z$). It can be verified that all the possible combinations of these two loss functions in $\mathfrak{L}$ are continuous and convex (even though not differentiable everywhere) with respect to vector $\mathbf{w}$.

Choosing the weighting factor γ_j is not straightforward. On the one hand, a key property of backfilling algorithms is that jobs of small area (small p, small q) are easier to backfill. Underpredicting small jobs can therefore mean delaying a reservation, and one should therefore use a weighting factor which *decreases* with p and q. On the other hand, as we consider systems with no preemption, once a large

γ_j	Interpretation
1	Constant weight.
$5 + \log(\frac{q_j}{p_j})$	Short jobs with large resource request should be well-predicted.
$5 + \log(\frac{p_j}{q_j})$	Long jobs with small resource request should be well-predicted.
$11 + \log(\frac{1}{q_j.p_j})$	Jobs of small "area" should be well-predicted.
$\log(q_j.p_j)$	Jobs of large "area" should be well-predicted.

Table 7.3. Weighting factors considered for training the model. The constants are chosen to ensure positivity of the weights with typical running times and resource requests in the HPC domain. Logarithms are used to alleviate the high range produced by ratios.

(large p, large $q \approx m$) job is started, almost no resources remain available. Thus, jobs in the queue are doomed to wait a long time. It follows that predicting jobs of large area correctly should be beneficial, and one should therefore use a weighting factor which *increases* with p and q. To account for these various elements, we explore four different possibilities along with a constant weighting factor, all of which are shown in Table 7.3.

Finally, for each choice of two basis loss function L_u and L_o along with weighting scheme γ_j, the regression model (*i.e.* the vector $\mathbf{w}$) is learned on an on-line training/testing set by minimizing the cumulative loss through the Normalized Adaptive Gradient [Ros+13] algorithm (NAG), a variant of the classical Stochastic Gradient Descent [Bot04]. The NAG algorithm poses the advantage of being robust to adversarial[3] scaling of the features. Robustness to feature scaling is a requirement of our problem because some features are difficult or impossible to normalize (*e.g.* BREAK TIME is unbounded). Section 7.5 describes the data sets retained as well as the values of the parameters considered for the search. Note that when $\gamma_j = 1$ and $L_u(z) = L_o(z) = z^2$, one is just considering a standard squared loss regression problem, learned in an on-line manner.

7.4 Scheduling

7.4.1 The EASY algorithm

As mentioned in Section 2.2, we target *EASY* because of it is broadly used and it has well-established performances.

Tsafrir *et al.* proposed in [Tsa+07] a slightly modified version of EASY, called EASY-SJBF, which performed better with running time predictions than the standard version. During the phase when the algorithm determines the candidate jobs to be backfilled, the jobs are sorted by increasing predicted running times instead of

[3]The notion of *adversary* simply means that the scaling can be arbitrary. See [CBL06] for a comprehensive study.

considering the FCFS order. They argue that this way, more jobs will be backfilled and thus, the overall performances will increase.

7.4.2 Correction mechanism

In this section, we are interested in the following question: *what happens to the schedule when the running times are mispredicted?*

The case of over-predicted running times is easy to handle by backfilling since the situation is the same as without learning where the users over-estimate the requested running times. In case of under-predicted running times, there are two points to consider. First, we should determine a new prediction for the remaining execution time and second, we have to check whether the correction does not disturb too much the scheduling algorithm. Both points are detailed as follows.

First point. We prefer to update the running times by some simple rules instead of computing again a prediction by the learning scheme, which gave a wrong value. Obviously, these updated running times remain bounded by the requested running times. These values are given by a correction algorithm. We consider the three following ones:

- REQUESTED TIME – Set the new prediction value to be $\widehat{p_j}$ (the user requested running time);
- INCREMENTAL – Use the corrective technique from [Tsa+07], *i.e.* increase at each correction by an fixed amount of time (1min, 5min, 15min, 30min, 1h, 2h, 5h, 10h, 20h, 50h, 100h);
- RECURSIVE DOUBLING – Increase the prediction value by the double of the elapsed running time.

Second point. Do backfilling variants handle the updated prediction? As the considered backfilling algorithms are on-line in nature, they adapt dynamically to the changes. However, notice that under-prediction with backfilling can lead to starvation. For instance, a large job will indefinitely wait for its required resources if under-predicted shorter jobs are systematically backfilled before. They will be launched before the large one, leading to an unbounded delay.

7.4.3 Objective Functions

In this chapter, all scheduling performances are measured with the average bounded slowdown (noted AVEbsld) function as it is a commonly admitted [Fei01] as a good objective function.

7.5 Experiments

Name	Year	# CPUs	# Jobs	Duration
KTH-SP2	1996	100	28k	11 Months
CTC-SP2	1996	338	77k	11 Months
SDSC-SP2	2000	128	59k	24 Months
SDSC-BLUE	2003	1,152	243k	32 Months
Curie	2012	80,640	312k	3 Months
Metacentrum	2013	3,356	495k	6 Months

Table 7.4. Workload logs used in the simulations.

7.5.1 Experiment objectives

The simulations we conducted aim at answering the following two questions:

1. Do the proposed predictive and corrective techniques improve existing scheduling algorithms?
2. Which prediction loss function, correction mechanism and backfilling variant work well together?

Previous studies have mainly focused on predicting running times, independently of the scheduling algorithms, using standard measures for the prediction error. In contrast, we aim here at predicting running times *for scheduling jobs with backfilling*, through a combination of appropriate loss functions, correction mechanisms and backfilling variants. The solutions we develop are thus closer to the problem of improving HPC systems. Moreover, identifying efficient combinations of prediction technique, correction mechanism and backfilling variants should provide insights into the behavior of backfilling algorithms when running times are unsure. In the rest of the chapter, we refer to such combinations as *heuristic triples*.

7.5.2 Description of the testbed

We make use in our study of a set of actual workload logs, described in Table 7.4. All these workload logs but Metacentrum are extracted from the Parallel Workload Archive [Fei+14]. Metacentrum is extracted from the personal website of Dalibor Klusáček[4]. They come from various HPC centers, correspond to highly different environments and have been selected for their high resource utilization, which challenges scheduling algorithms [FF05]. For each log, we run scheduling simulations using every possible heuristic triples: prediction technique, correction mechanism and backfilling variant.

All simulations are run using a fork of the open-source[5] batch scheduler simulator *pyss*. The source of this forked version is available on-line[6].

[4] `http://www.fi.muni.cz/$\sim$xklusac/`
[5] pyss - the Python Scheduler Simulator, available at `http://code.google.com/p/pyss/`
[6] `http://github.com/freuk/predictsim`

Parameter	Possible Values
L_u	$z \mapsto z^2$, $z \mapsto z$
L_o	$z \mapsto z^2$, $z \mapsto z$
γ_j	See Table 7.3 (5 values)

Table 7.5. Considered parameter values of the loss function. There are three effective parameters, for a total of 20 combinations.

All prediction techniques based on the different loss functions and weighting schemes introduced in Section 7.3 are considered here, in conjunction, for comparison purposes, with the actual running time p_j, denoted as CLAIRVOYANT, the user requested time $\widetilde{p_j}$, denoted as REQUESTED TIME and the average of the two previous running times for the jobs of user k, denoted as $AVE_2^{(k)}(p)$. For correction, we rely on the three correction techniques presented in Subsection 7.4.2: REQUESTED TIME, INCREMENTAL and RECURSIVE DOUBLING. Lastly, we rely on the two backfilling algorithms presented in Subsection 7.4.1 , namely EASY and EASY-SJBF.

Notice that the case where REQUESTED TIME is used as prediction technique and EASY as the backfilling variant corresponds to the standard EASY backfilling algorithm. Similarly, the case where INCREMENTAL correction method is used with the $AVE_2^{(k)}(p)$ prediction technique and the EASY-SJBF backfilling variant correspond to the EASY++ algorithm introduced by Tsafrir *et al.* [Tsa+07].

As it is reasonable to expect that scheduling performances due to a loss function (and therefore, a learned model) are dependent on both the backfilling variant and correction mechanism, we evaluate all of them together. This induces a higher complexity and a high number of simulations. For each workload log, the experimental campaign runs 128 simulations. As it is impossible to present all of them in detail, we invite the reader to look at our repository[6].

The experiment campaign contains significantly more heuristic triples than workload logs, and this raises a multiple hypothesis testing problem. Therefore, one should approach the analysis of the results using sound statistical practices.

Subsection 7.5.3 outlines the best heuristic triple. Afterwards, Subsection 7.5.4 contains an analysis of the predictions made.

7.5.3 Which heuristic triple is prevalent?

Raw results

As shown in Table 7.6 which displays the AVEbsld scores for the different approaches, the results seem promising as they tend to favor approaches based on learning (the CLAIRVOYANT results are reported for comparison purposes only and correspond to an upper bound of what one can expect). However, one should not conclude too hastily, as even though the best approach is always obtained using a

Table 7.6. Overview of the AVEbsld for each simulations. For predictive techniques, only the best and the worst AVEbsld are given. The best non-clairvoyant heuristic triples are outlined in bold.

	Clairvoyant EASY				EASY with Learning Techniques	
Trace	FCFS	SJBF	EASY	EASY++	FCFS	SJBF
KTH-SP2	71.7	49.8	92.6	63.5	62.6 - 93.2	**51.4** - 74.5
CTC-SP2	37.2	17.6	49.6	85.8	25.5 - 163.5	**16.3** - 134.7
SDSC-SP2	70.5	56.8	87.9	79.4	70.9 - 102.3	**69.7** - 194.8
SDSC-BLUE	30.6	13.2	36.5	21.0	16.5 - 48.0	**12.6** - 47.8
Curie	69.9	12.1	202.1	193.5	26.3 - 9348.8	**24.3** - 4010
Metacentrum	81.7	67.2	97.6	87.2	86.3 - 98.1	**81.5** - 89.8

predictive-corrective approach (corresponding to the columns EASY with Learning Techniques in Table 7.6), it is not clear which heuristic triple is prevalent *a priori*.

In particular we observe that the SJBF variant introduced by [Tsa+07] is rather efficient at leveraging accurate values of the running times, as the CLAIRVOYANT EASY-SJBF algorithm almost always outperforms its competitors.

Correlation between logs

A key part of the analysis of predictive techniques is to see how performances correlate between different systems.

Figure 7.2 shows the AVEbsld between the MetaCentrum and SDSC-BLUE logs, meant to illustrate the irregularity of performances between logs. We can observe that CLAIRVOYANT EASY-SJBF is the best in both logs, but there is no clear correlation between all the heuristic triples.

The correlation coefficient is measured here with the Pearson's Correlation coefficient, which is computed for each couple of logs. With a mean of 0.26 (min: 0.01, max: 0.80), this coefficient is low. This means that it is not possible to know from the result on one log if a heuristic triple will perform well on another logs. However, one can still try and learn an appropriate heuristic triple from existing systems, as described below.

Triple selection

We consider here a *leave-one-out cross validation* process in which 5 logs are (alternatively) used to select the best triple, the performance of which is evaluated on the 6th log. The idea is to assess whether one can select a good heuristic triple from existing logs. The experiment is repeated six times (for the six logs) and the results are averaged over the six repetitions. The best heuristic triple is the one that optimizes the sum of the AVEbsld on the 5 logs. Table 7.7 displays the obtained results. As one can note, the results obtained with this selection process, denoted C-V (for cross-validation) heuristic triple, significantly outperforms the EASY and EASY++ approaches on all workloads except SDSC-BLUE.

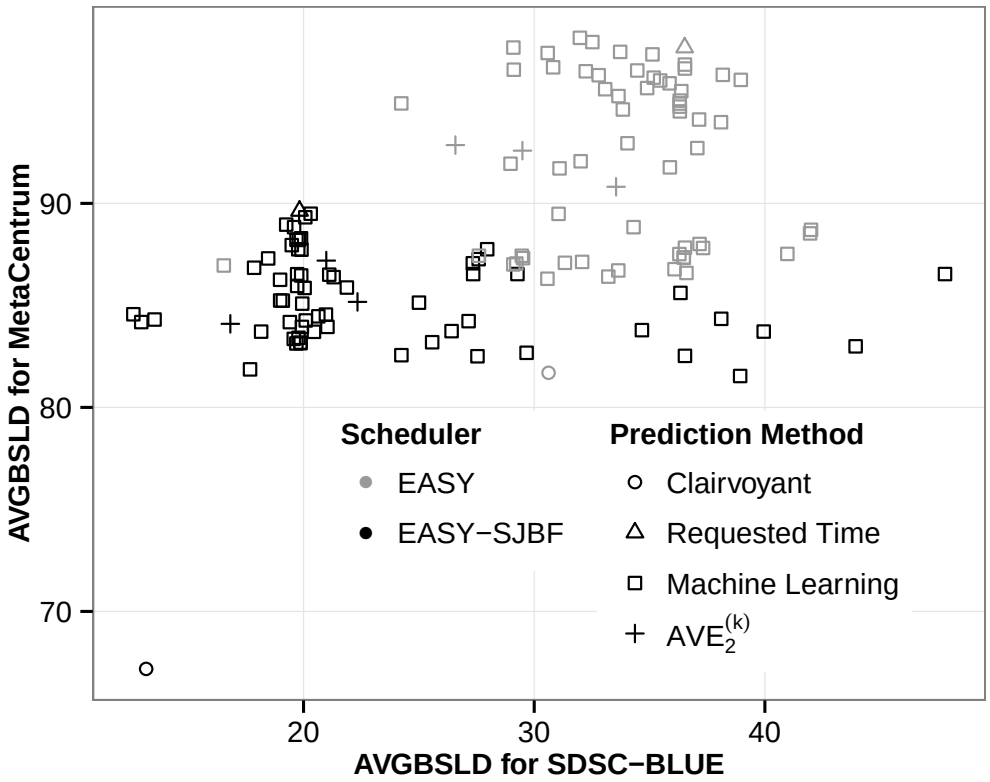


Figure 7.2. Scatter plot of heuristic's relative performance between the MetaCentrum and SDSC-BLUE logs.

Log	C-V Heuristic triple	EASY	EASY++
KTH-SP2	**51.4** (44%)	92.6	63.5 (31%)
CTC-SP2	**20.5** (59%)	49.6	85.8 (-72%)
SDSC-SP2	**75.0** (15%)	87.9	79.4 (10%)
SDSC-BLUE	34.7 (05%)	36.5	**21.0** (42%)
Curie	**27.9** (86%)	202.1	193.5 (04%)
Metacentrum	**84.2** (14%)	97.6	87.2 (11%)

Table 7.7. AVEbsld performance of the heuristic triples resulting from cross validation. Values in parenthesis show the AVEbsld reduction obtained respective to EASY.

Even more interestingly, it turns out that the best heuristic triple on all logs using the selection method above is the same[7] and corresponds to the following setting:

Prediction Technique : Regression function described in Section 7.3 with the loss function:

$$\mathfrak{L}(\mathbf{x}_j, f(\mathbf{x}_j), p_j) = \begin{cases} \log(r_j.p_j).(f(\mathbf{x}_j) - p_j)^2 & \text{if } f(\mathbf{x}_j) \geq p_j \\ \log(r_j.p_j).(p_j - f(\mathbf{x}_j)) & \text{if } f(\mathbf{x}_j) < p_j \end{cases} \tag{7.3}$$

Correction mechanism : INCREMENTAL

backfilling variant : EASY-SJBF

Summary

We have shown here that one can learn an appropriate heuristic triple from existing logs. This heuristic triple yields better scheduling performances than EASY and EASY++. Furthermore, on the workloads considered, a heuristic triple singles out as it is the one always selected. This heuristic triple obtains an average AVEbsld reduction of 28% compared to EASY and 11% compared to EASY++, and can reduce the AVEbsld by 86% compared to EASY (on the Curie workload for example). This triple uses the INCREMENTAL correction technique and the SJBF queue ordering from [Tsa+07], as well as a machine learning-based approach with custom loss functions (7.3). We call this loss function E-Loss (for EASY-Loss) and briefly discuss its behavior in the next Subsection.

7.5.4 Prediction analysis

The first question one usually asks after having used a predictive technique is, *what is the prediction accuracy*? Experimental results outline that while prediction performance is important, choosing the right loss for the prediction is even more critical. This is observed in Table 7.8 which shows the prediction errors of both the $AVE_2^{(k)}(p)$ prediction technique and our E-Loss based approach.

Prediction Technique	MAE	Mean E-Loss
$AVE_2^{(k)}(p)$	5217	10.2×10^8
E-Loss Learning	6762	2.35×10^5

Table 7.8. MAE and E-Loss for different prediction techniques. All values are in seconds.

One can see from these values that while the $AVE_2^{(k)}(p)$ performs well with respect to the Mean Average Error (MAE), its performance on the E-Loss is quite poor.

Equation (7.3) shows that the E-Loss is an asymmetrical loss function, with a linear branch for under-prediction and a squared branch for over-prediction. Therefore this loss function discourages over-prediction. Additionally, the E-Loss

[7] with one exception: the C-V Heuristic selected for SDSC-SP2 uses the REQUESTED TIME correction mechanism. This could account for it's degraded performance.

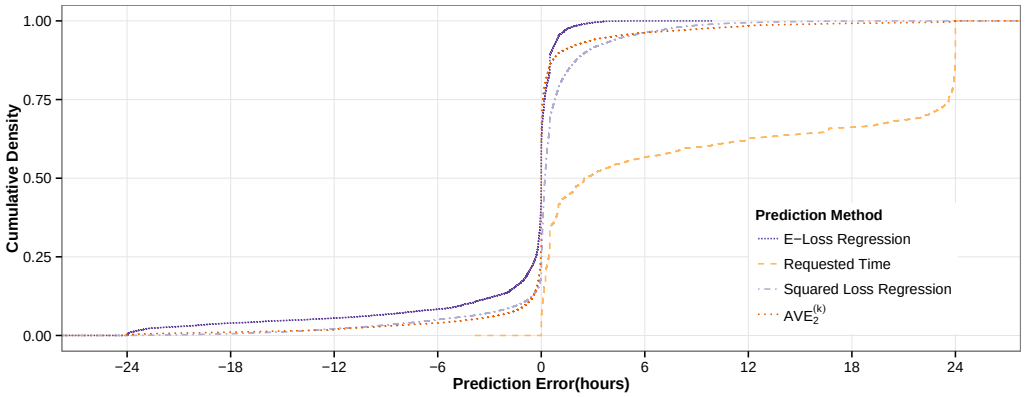


Figure 7.3. Experimental cumulative distribution functions of prediction errors obtained using the Curie log.

uses a weighting factor that increases with the size of jobs in terms of p and q. A helpful visualization for understanding how the E-loss behaves in practice is the empirical cumulative distribution function (ECDF) of the prediction errors produced by the resulting machine learning model. Figure 7.3 shows the ECDFs of such prediction errors for main prediction techniques. From this figure, one can see the behavior of the E-loss with respect to that of the standard squared loss. The E-loss ECDF is shifted to the left, which means that more under-prediction errors are indeed made than with standard regression. This is coherent with intuition gleaned from the analysis form of the loss function.

Finally, Figure 7.4 shows the ECDF of the values that were predicted. On this graph, we see that in order to generalize well with respect to the E-Loss, the learning model ends up being strongly biased towards small predictions. This displacement suggests that there might be a beneficial effect to backfilling jobs very aggressively when using EASY-SJBF.

7.5.5 Discussion

As mentioned above, the proposed approach outperforms EASY++ with a **11%** of reduction in average AVEbsld, and has a reduction of **28%** in average AVEbsld when compared to EASY. This result is obtained by changing the prediction technique of EASY++ to one that uses a custom loss function that we refer to as E-loss. We have furthermore observed that on each log, roughly 0.1% of jobs have extremely high values of bounded slowdowns. Such a behavior is obtained with every heuristic triple based on $AVE_2^{(k)}(p)$ or MACHINE LEARNING prediction techniques. Extreme values seem to be a shortcoming of incorporating predictions without a mechanism for dealing with extreme prediction failures. Moreover, because such failures are often due to jobs that do not run properly, we are confronted here with an evaluation problem, as the cost of such events could be incurred on the user rather than the system. New performance evaluation measures are needed to deal

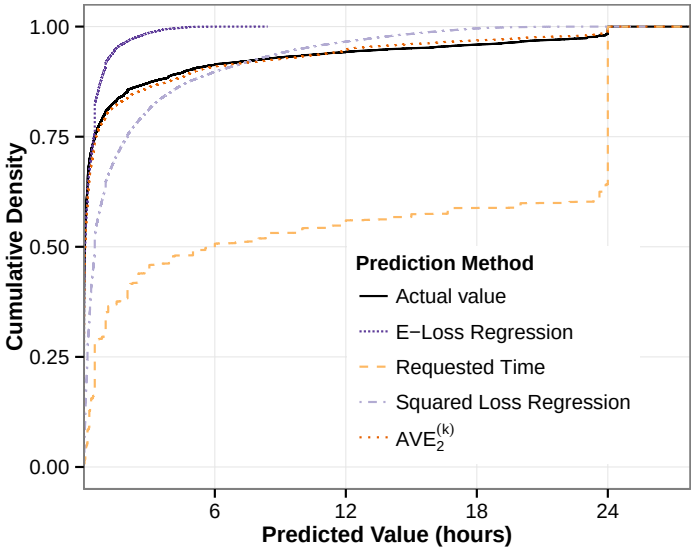


Figure 7.4. Experimental cumulative distribution functions of predicted values obtained using the Curie log.

with such problems, especially for schedulers without no-starvation guarantees (as other authors already suggested [FF05]).

7.6 Conclusion

The purpose of this work was to investigate whether the use of learning techniques on the job running times is worth for improving the scheduling algorithms. We proposed a new cost function for prediction and run simulations based on actual workload logs for the most popular variants of backfilling. The results clearly show that this approach is very useful, as they reduce the average bounded slowdown by a factor of 28% compared to EASY. Moreover, the proposed approach may be extended easily to other scheduling policies.

Acknowledgment

We would like to warmly thank Yiannis Georgiou for his unfailing support. We are also indebted to Krzysztof Rzadca for helpful discussions and support. We thank gracefully the contributors of the Parallel Workloads Archive, Victor Hazlewood (SDSC SP2), Travis Earheart and Nancy Wilkins-Diehr (SDSC Blue), Lars Malinowsky (KTH SP2), Dan Dwyer and Steve Hotovy (CTC SP2), Joseph Emeras (CEA Curie), and of course Dror Feitelson. The Metacentrum workload log was graciously provided by the Czech National Grid Infrastructure MetaCentrum. The work is partially supported by the ANR project MOEBUS.

Conclusions and future research directions

8

8.1 Conclusions

Resource and Job Management Systems (RJMS) are an essential bulk for the HPC community in order to reach exaflop systems. Exascale centers will open new paths for researchers by computing bigger computations and more accurate simulations. Energy consumption and scheduling performances are key components to such future centers. Through various studies, we have shown that we can efficiently control the power and energy consumption and employ users behaviors to improve scheduling performances.

More precisely, we first developed methodologies to simulate or emulate underlying platforms. We also show how to measure power and energy within a large scale platform and implement this methodology in a widely used RJMS (SLURM). Thanks to these tools, we were able to develop several new mechanisms to test if users can help reaching exascale systems.

Second, energy consumption of such systems is one of the challenges to build exascale systems, thus, we proposed several mechanisms to control power and energy consumption. To further reduce energy consumption, energy consumption of jobs should be optimized. Thus, we developed a technique that benefit from users behaviors to incite them to improve the energy efficiency of their jobs.

Finally, we designed an online machine learning algorithm to improve scheduling performances. The machine learning algorithm learns from past users behaviors to predict the running times of jobs. This predicted running time is not computed to be the more accurate but to be the value that will give the best scheduling possible.

All these empirical studies stress that energy and power can be controlled. Moreover, to reduce energy consumption while increasing (or at least keeping) overall performances in huge platforms is a tedious task because the algorithms have to take into account information about the users, the platform, and the jobs. In this thesis, we proposed algorithms that are fast and improve actual algorithms by connecting together all these elements.

8.2 Future works

This thesis, through its contributions, is a step towards exascale platforms. However, the future clusters will require to optimize more objectives and support more constraints than today. How to develop new allocation and scheduling algorithms supporting these features in an environment where $O(n)$ is the maximum possible complexity? A possible idea is to obtain a first schedule thanks to a fast greedy

algorithm and then improves it using algorithms. However, such an approach suffers from a big drawback: the future is composed of too many uncertainties. We propose to tackle this by using machine learning algorithms coupled with the scheduling algorithm. We have shown in our work that this is possible. It should be possible to support even more features.

HPC systems are very dynamic systems in nature. Thus, the task to evaluate new RJMS scheduling algorithms is very tedious. The experience acquired by the experiments done in this thesis, motivate us to not only focus on algorithm design, but also on their implementations. We believe that using actual RJMSs to experiment new mechanisms is one step forward to test their performance. In this context, simulators and emulators are needed to perform the experiments on actual RJMSs. We presented some first steps towards this idea. However, the simulator needs to be completed and compared to real runs. Then, it needs to support more features, like being able to simulate energy consumption.

An inherent limitation of our experiments is the kind of workloads used in the experiments. We either use a real workload trace or a synthetic model of a real workload trace. These kind of workloads are specific to one cluster at a given time and do not take into account how users behave with HPC systems. For example, if a job run for only a few seconds, there is a high probability that a similar job would start just after (as the user observe that her job crashes, she starts a new one with the bug corrected). We can even go further and ask if users submit jobs when they need or when their results (the end of the job) will finish *soon enough*. In other words, if we double the size of a given cluster, does its utilization decrease? As the users implicitely decide on the load of the machine (through job submissions), determining the condition of the submission of jobs may help us to develop a better model to assess the scheduling algorithms.

The powercapping algorithm can be adapted by considering the real-time power consumption measures of the nodes, instead of considering the static values defined during the initialization phase. Moreover, we will consider to dynamically change the CPU frequencies while the jobs are running, this will allow nodes to adjust the power consumption instantly whenever it is needed. This will eventually result in faster power decrease when a powercap period is approaching and lower jobs' turnaround time after a powercap period is over. A finer control over the power consumption on a power constrained environment will improve the overall performances.

We observed with the energycap algorithm on chapter 5 that reducing the energy available for the cluster by a ratio does not mean to reduce the overall performance of the cluster by the same ratio. Further research should be done in this direction. It looks like the effect developed in [Pat+13] where they argue that buying a bigger cluster and then limit the power consumption is monetary worth it. Do we observe the same effect?

Evenmore, this energycap algorithm can be extended by dynamically adjusting the energy available in the cluster based on the dynamic cost of electricity. With this

mechanism, the cluster electricity cost will be fixed for a period of time. A similar work can be done by adjusting the energy available to the green energy production of the grid (or by a local equipment) and thus set the ecological footprint.

From discussion with cluster's owners, administrators, and users, we observed that there is not a unique way to control and reduce the energy consumption. Thus, creating a new algorithm for each new system will be too tedious. A possible future work is to develop a framework that can be adapted to the use case of each specific cluster. For example, many clusters can use switch-off based mechanisms. Most HPC clusters are built to always work at maximum speed. Switching off a node is considered neither in the software nor in the hardware. In the software part that interest us, the RJMS, using switch off smartly requires deep changes. It needs to determine a trade-off between switching on some nodes to start a job, launching the job on another part of the cluster, or starting the job later. All of these actions have a non-obvious impact on the global scheduling and thus on the performances.

Scheduling using machine learning is a promising approach as it shows in our study impressive results. We think that the main strength of using this kind of algorithm is their dynamic adaptability: they adapt to the global environment but also to the current happening events. Thus, a next step can be to sort the list of queued job thanks to a preference learning algorithm. Also, machine learning can help in predicting job bursts, machine failures, best DVFS values, etc. More than predicting these values, we have to develop ways to take into account the statistical predictions with the scheduling algorithm.

Finally, our study on taking advantage of users behaviors and controlling power and energy in large scale system can be extended to fit the needs for the constant hardware and software evolutions. The continuous study and evolution of resource and job management systems are indispensable to take the most of HPC systems.

References

A

Publications

[Gau+] Eric Gaussier, David Glesser, Valentin Reis, and Denis Trystram. “Improving Backfilling by Using Machine Learning to Predict Running Times”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SuperComputing 2015 (cit. on pp. 4, 49, 74, 106).

[Geo+a] Y. Georgiou, D. Glesser, K. Rzadca, and D. Trystram. “A Scheduler-Level Incentive Mechanism for Energy Efficiency in HPC”. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (cit. on pp. 4, 15, 56, 106).

[Geo+b] Y. Georgiou, D. Glesser, and D. Trystram. “Adaptive Resource and Job Management for Limited Power Consumption”. In: 2015 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW) (cit. on pp. 4, 15, 25, 46, 106).

[Geo+c] Yiannis Georgiou, Thomas Cadeau, David Glesser, et al. “Energy Accounting and Control with SLURM Resource and Job Management System”. In: *Distributed Computing and Networking*. Lecture Notes in Computer Science 8314 (cit. on pp. 4, 15, 18, 21, 36, 57, 106).

Bibliography

[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 (cit. on p. 5).

[Aik+11] D. Aikema, C. Kiddle, and R. Simmonds. “Energy-cost-aware scheduling of HPC workloads”. In: *WoWMoM*. 2011 (cit. on p. 26).

[Alb10] Susanne Albers. “Energy-efficient algorithms”. In: *Communications of the ACM* (2010) (cit. on p. 58).

[Ass+12] MarcosDias Assunção, Jean-Patrick Gelas, Laurent Lefèvre, and Anne-Cécile Orgerie. “The Green Grid’5000: Instrumenting and Using a Grid with Energy Sensors”. In: *Remote Instrumentation for eScience and Related Aspects*. 2012 (cit. on p. 57).

[Bal+13] 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. 2013 (cit. on p. 50).

[Bam+16] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. “Energy-Aware Scheduling for Real-Time Systems: A Survey”. In: *Transactions on Embedded Computing Systems (TECS)* (2016) (cit. on p. 46).

[Ber+95] Herman JC Berendsen, David van der Spoel, and Rudi van Drunen. “GROMACS: A message-passing parallel molecular dynamics implementation”. In: *Computer Physics Communications* (1995) (cit. on p. 35).

[BH07] Luiz André Barroso and Urs Hölzle. “The case for energy-proportional computing”. In: *IEEE Computer* (2007) (cit. on p. 47).

[BM12] Robert Basmadjian and Hermann de Meer. “Evaluating and modeling power consumption of multi-core processors”. In: *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*. IEEE. 2012 (cit. on p. 57).

[Bol+06] Raphaël Bolze, Franck Cappello, Eddy Caron, et al. “Grid’5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed”. In: *IJHPCA* (2006) (cit. on p. 11).

[Bot04] Léon Bottou. “Stochastic learning”. In: *Advanced lectures on machine learning*. 2004 (cit. on p. 81).

[Cap+05] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, et al. “A batch scheduler with high level components”. In: *CCGrid 2005*. 2005 (cit. on p. 5).

[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* (2014) (cit. on pp. 17, 49).

[CB01a] Walfredo Cirne and Francine Berman. “A Comprehensive Model of the Supercomputer Workload”. In: *4th Workshop on Workload Characterization*. Dec. 2001, pp. 140–148 (cit. on p. 12).

[CB01b] Walfredo Cirne and Francine Berman. “A Model for Moldable Supercomputer Jobs”. In: *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS-01)*. Los Alamitos, CA: IEEE Computer Society, 2001, pp. 59–59 (cit. on p. 12).

[CBL06] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006 (cit. on p. 81).

[Cha+99] Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, et al. “Benchmarks and Standards for the Evaluation of Parallel Job Schedulers”. In: *JSSPP’99*. 1999 (cit. on p. 12).

[Dan14] Daniel Hackenberg et al. “HDEEM: High Definition Energy Efficiency Monitoring”. In: *2nd International Workshop on Energy Efficient Supercomputing*. 2014 (cit. on p. 57).

[Dem+07] Erik D. Demaine, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, Amin S. Sayedi-Roshkhar, and Morteza Zadimoghaddam. “Scheduling to Minimize Gaps and Power Consumption”. In: *SPAA*. 2007 (cit. on p. 26).

[Dem+12] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. “Communication-optimal parallel and sequential QR and LU factorizations”. In: *SIAM Journal on Scientific Computing* (2012) (cit. on pp. 55, 58).

[DF99] Allen B. Downey and Dror G. Feitelson. “The elusive goal of workload characterization”. In: *SIGMETRICS Perform. Eval. Rev.* (1999) (cit. on p. 12).

[Don+11] Jack Dongarra, Pete Beckman, Terry Moore, et al. “The International Exascale Software Project Roadmap”. In: *Int. J. High Perform. Comput. Appl.* (2011) (cit. on pp. 1, 7).

[Dua+09] Rubing Duan, F. Nadeem, Jie Wang, et al. “A Hybrid Intelligent Method for Performance Modeling and Prediction of Workflow Activities in Grids”. In: *Cluster Computing and the Grid*. 2009 (cit. on p. 76).

[Dut+16] 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. 49).

[Eme+14] Joseph Emeras, Vinicius Pinheiro, Krzysztof Rzadca, and Denis Trystram. “OStrich: Fair Scheduling for Multiple Submissions”. In: *PPAM, Proc.* 2014 (cit. on p. 58).

[Eti+10a] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. “BSLD threshold driven power management policy for HPC centers”. In: *IPDPSW*. 2010 (cit. on pp. 26, 33, 37).

[Eti+10b] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. “Optimizing job performance under a given power constraint in HPC centers”. In: *IGCC*. 2010 (cit. on p. 26).

[Eti+12a] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. “Parallel job scheduling for power constrained HPC systems”. In: *Parallel Computing* (2012) (cit. on pp. 26, 46).

[Eti+12b] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. “Understanding the future of energy-performance trade-off via DVFS in HPC environments”. In: *Journal of Parallel and Distributed Computing* (2012) (cit. on p. 26).

[Fan+07] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. “Power provisioning for a warehouse-sized computer”. In: *SIGARCH* (2007) (cit. on p. 26).

[FD] LSF (Load Sharing Facility) Features and Documentation. http://www.platform.com/workload-management/high-performance-computing (cit. on p. 5).

[Fei+14] Dror G. Feitelson, Dan Tsafrir, and David Krakov. “Experience with using the Parallel Workloads Archive”. In: *Journal of Parallel and Distributed Computing* (2014) (cit. on pp. 50, 77, 83).

[Fei01] Dror G. Feitelson. “Metrics for Parallel Job Scheduling and Their Convergence”. In: *Job Scheduling Strategies for Parallel Processing (JSSPP)*. 2001 (cit. on pp. 8, 82).

[Fei03] Dror G. Feitelson. “Metric and Workload Effects on Computer Systems Evaluation”. In: *IEEE Computer* (2003) (cit. on p. 12).

[FF05] Eitan Frachtenberg and Dror G. Feitelson. “Pitfalls in parallel job scheduling evaluation”. In: *Job Scheduling Strategies for Parallel Processing (JSSPP)*. 2005 (cit. on pp. 8, 11, 12, 83, 89).

[Fol+99] Mike Folk, Albert Cheng, and Kim Yates. “HDF5: A file format and i/o library for high performance computing applications”. In: *Proceedings of Supercomputing’99 (CD-ROM)*. Portland, OR: ACM SIGARCH and IEEE, Nov. 1999 (cit. on p. 21).

[Fre+07] Vincent W. Freeh, David K. Lowenthal, Feng Pan, et al. “Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications”. In: *TPDS* (2007) (cit. on pp. 26, 37).

[FS07] Eitan Frachtenberg and Uwe Schwiegelshohn. “New Challenges of Parallel Job Scheduling”. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Eitan Frachtenberg and Uwe Schwiegelshohn. Springer Verlag, 2007 (cit. on p. 12).

[Gan+09] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, Jeffrey O. Kephart, and Charles Lefurgy. “Power capping via forced idleness”. In: *WEED*. 2009 (cit. on p. 26).

[Geo+15] Y. Georgiou, D. Glesser, K. Rzadca, and D. Trystram. “A Scheduler-Level Incentive Mechanism for Energy Efficiency in HPC”. In: *Cluster, Cloud and Grid Computing (CCGrid)*. 2015 (cit. on p. 46).

[Geo10] Yiannis Georgiou. “Contributions for Resource and Job Management in High Performance Computing”. PhD thesis. 2010 (cit. on pp. 5, 11).

[Ger+14] Guilherme Arthur Geronimo, Jorge Werner, Rafael Weingartner, Carlos Becker Westphall, and Carla Merkle Westphall. “Provisioning, Resource Allocation, and DVFS in Green Clouds”. In: *IJANS* (2014) (cit. on p. 26).

[GH12] Yiannis Georgiou and Matthieu Hautreux. “Evaluating scalability and efficiency of the Resource and Job Management System on large HPC Clusters”. In: *JSSPP*. 2012 (cit. on pp. 13, 39, 65, 68).

[Gho+11] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, et al. “Dominant Resource Fairness: Fair Allocation of Multiple Resource Types.” In: *NSDI*. 2011 (cit. on pp. 58, 59).

[Gib97] Richard Gibbons. “A historical application profiler for use by parallel schedulers”. In: *Job Scheduling Strategies for Parallel Processing*. 1997 (cit. on p. 75).

[Gio13] Giorgio Luigi Valentini et al. “An overview of energy efficiency techniques in cluster computing systems”. In: *Cluster Computing* (2013) (cit. on pp. 57, 58).

[Gra66] Ronald L Graham. “Bounds for certain multiprocessing anomalies”. In: *Bell System Technical Journal* (1966) (cit. on p. 5).

[Hik+08] Junichi Hikita, Akio Hirano, and Hiroshi Nakashima. “Saving 200kw and $200 k/year by power-aware job/machine scheduling”. In: *International Parallel & Distributed Processing Symposium (IPDPS)*. 2008 (cit. on pp. 26, 47).

[Imb] *https://software.intel.com/en-us/articles/intel-mpi-benchmarks* (cit. on p. 35).

[Int] Intel. *Intelligent Platform Management Interface Specification v2.0* (cit. on pp. 20, 56).

[Jac+01] David Jackson, Quinn Snell, and Mark Clement. “Core algorithms of the Maui scheduler”. In: *JSSPP, Proc.* 2001 (cit. on pp. 58, 59).

[JW+13] Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. “Multiresource Allocation: Fairness-efficiency Tradeoffs in a Unifying Framework”. In: *IEEE/ACM Trans. Netw.* (2013) (cit. on p. 59).

[Kap+05] Nandini Kappiah, Vincent W Freeh, and David K Lowenthal. “Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs”. In: *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. 2005 (cit. on p. 58).

[KC03] S.D. Kleban and S.H. Clearwater. “Fair share on high performance computing systems: what does fair really mean?” In: *CCGrid, Proc.* 2003 (cit. on p. 58).

[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 (cit. on p. 46).

[Kim+08] H. Kimura, M. Sato, T. Imada, and Y. Hotta. “Runtime DVFS control with instrumented Code in power-scalable cluster system”. In: *Cluster*. 2008 (cit. on p. 26).

[Kim+12] Shin-gyu Kim, Chanho Choi, Hyeonsang Eom, H.Y. Yeom, and Huichung Byun. “Energy-Centric DVFS Controling Method for Multi-core Platforms”. In: *SCC*. 2012 (cit. on p. 25).

[Kon+10] Derrick Kondo, Bahman Javadi, Alexandru Iosup, and Dick Epema. “The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems”. In: *Cluster Computing and the Grid, IEEE International Symposium on* (2010) (cit. on p. 15).

[KR13] Dalibor Klusacek and Hana Rudova. “Performance and Fairness for Users in Parallel Job Scheduling”. In: *JSSPP, Proc.* 2013 (cit. on p. 58).

[KR14] Dalibor Klusaček and Hana Rudova. “Multi-Resource Aware Fairsharing for Heterogeneous Systems”. In: *JSSPP, Proc.* 2014 (cit. on p. 59).

[Len+13] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, et al. “Gpuwattch: Enabling energy optimizations in gpgpus”. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. 2013 (cit. on p. 57).

[Lif95] David A Lifka. “The anl/ibm sp scheduling system”. In: *Job Scheduling Strategies for Parallel Processing*. 1995 (cit. on p. 6).

[Lin] *http://www.netlib.org/linpack/* (cit. on p. 35).

[Lit+88] Michael J Litzkow, Miron Livny, and Matt W Mutka. “Condor-a hunter of idle workstations”. In: *Distributed Computing Systems, 1988., 8th International Conference on*. 1988 (cit. on p. 5).

[LS05] Barry Lawson and Evgenia Smirni. “Power-aware resource allocation in high-end systems via online simulation”. In: *SC*. 2005 (cit. on p. 26).

[McC95] John D McCalpin. “A survey of memory bandwidth and machine balance in current high performance computers”. In: *IEEE TCCA Newsletter* (1995) (cit. on p. 35).

[MF01a] 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* (2001) (cit. on p. 6).

[MF01b] A.W. Mu'alem and D.G. Feitelson. "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling". In: *TPDS* (2001) (cit. on pp. 29, 59).

[MF10] A. Matsunaga and J. Fortes. "On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications". In: *Cluster, Cloud and Grid Computing*. 2010 (cit. on p. 76).

[Mit14] Sparsh Mittal. "Power Management Techniques for Data Centers: A Survey". In: *arXiv preprint arXiv:1404.6681* (2014) (cit. on p. 57).

[Moo+05] Justin D. Moore, Jeffrey S. Chase, Parthasarathy Ranganathan, and Ratnesh K. Sharma. "Making Scheduling" Cool": Temperature-Aware Workload Placement in Data Centers." In: *USENIX annual technical conference, General Track*. 2005 (cit. on p. 26).

[MR98] C.L. Mendes and D.A. Reed. "Integrated compilation and scalability analysis for parallel systems". In: *Parallel Architectures and Compilation Techniques*. 1998 (cit. on p. 76).

[MV15] P. Murali and S. Vadhiyar. "Metascheduling of HPC Jobs in Day-Ahead Electricity Markets". In: *High Performance Computing (HiPC)*. 2015 (cit. on p. 46).

[NF08] Avi Nissimov and Dror G. Feitelson. "Probabilistic Backfilling". In: *Job Scheduling Strategies for Parallel Processing*. 2008 (cit. on p. 76).

[Nis06] Avi Nissimov. "Locality and its usage in parallel job runtime distribution modeling using HMM". MA thesis. The Hebrew University, 2006 (cit. on pp. 76, 77).

[Nit+04] Bill Nitzberg, Jennifer M Schopf, and James Patton Jones. "PBS Pro: Grid computing and scheduling attributes". In: *Grid resource management*. 2004 (cit. on p. 5).

[Org+08] 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: *International Conference on Parallel and Distributed Systems (ICPADS)*. 2008 (cit. on p. 47).

[Pat+13] Tapasya Patki, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis R De Supinski. "Exploring hardware overprovisioning in power-constrained, high performance computing". In: *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM. 2013, pp. 173–182 (cit. on p. 92).

[Pat+15] Tapasya Patki, David K. Lowenthal, Anjana Sasidharan, et al. “Practical Resource Management in Power-Constrained, High Performance Computing”. In: *High-Performance Parallel and Distributed Computing (HPDC)*. 2015 (cit. on p. 46).

[PC11] Jean-Marc Pierson and Henri Casanova. “On the utility of dvfs for power-aware job placement in clusters”. In: *Euro-Par*. 2011 (cit. on p. 27).

[Pys] Pyss. *https://code.google.com/p/pyss/* (cit. on p. 63).

[Rab89] Lawrence Rabiner. “A tutorial on hidden Markov models and selected applications in speech recognition”. In: *Proceedings of the IEEE* (1989) (cit. on p. 76).

[Red+12] Sherief Reda, Ryan Cochran, and Ayse K. Coskun. “Adaptive power capping for servers with multithreaded workloads”. In: *IEEE Micro* (2012) (cit. on p. 26).

[Ros+13] Stéphane Ross, Paul Mineiro, and John Langford. “Normalized Online Learning”. In: *Uncertainty in Artificial Intelligence* (2013) (cit. on pp. 78, 81).

[Ros+14] Francois Rossigneux, Jean-Patrick Gelas, Laurent Lefevre, and Marcos Dias de Assuncao. “A Generic and Extensible Framework for Monitoring Energy Consumption of OpenStack Clouds”. In: *arXiv preprint arXiv:1408.6328* (2014) (cit. on p. 57).

[Rot+12] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. “Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge”. In: *IEEE Micro* (2012) (cit. on pp. 20, 57).

[Rou+11] B. Rountree, D.K. Lowenthal, M. Schulz, and B.R. De Supinski. “Practical performance prediction under Dynamic Voltage Frequency Scaling”. In: *IGCC*. 2011 (cit. on p. 26).

[Rou+12a] 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: *IPDPSW*. 2012 (cit. on p. 26).

[Rou+12b] 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 (IPDPSW)*. 2012 (cit. on p. 46).

[Sar+14] Osman Sarood, Akhil Langer, Abhishek Gupta, and Laxmikant Kale. “Maximizing throughput of overprovisioned hpc data centers under a strict power budget”. In: *SuperComputing (SC)*. 2014 (cit. on p. 46).

[Sch+99] Jennifer M. Schopf, Francine Berman, Jennifer M. Schopf, and Francine Berman. “Using Stochastic Intervals to Predict Application Behavior on Contended Resources”. In: *International Symposium on Parallel Architectures, Algorithms, and Networks*. 1999 (cit. on p. 76).

[SF07] Edi Shmueli and Dror G. Feitelson. “Uncovering the Effect of System Performance on User Behavior from Traces of Parallel Systems”. In: *MASCOTS*. 2007 (cit. on pp. 15, 41).

[SG06] Bianca Schroeder and Garth A. Gibson. “A large-scale study of failures in high-performance computing systems”. In: *Dependable Systems and Networks, International Conference on* (2006) (cit. on p. 15).

[SH11] Robert Schöne and Daniel Hackenberg. “On-line Analysis of Hardware Performance Events for Workload Characterization and Processor Frequency Scaling Decisions”. In: *ICPE*. 2011 (cit. on p. 26).

[Sha+03] Li Shang, Li-Shiuan Peh, and Niraj K. Jha. “Dynamic voltage scaling with links for power optimization of interconnection networks”. In: *HPCA-9*. 2003 (cit. on p. 26).

[Smi+04] Warren Smith, Ian Foster, and Valerie Taylor. “Predicting Application Run Times with Historical Information”. In: *Journal of Parallel and Distributed Computing* (2004) (cit. on p. 75).

[Sno+05] David C. Snowdon, Sergio Ruocco, and Gernot Heiser. “Power management and dynamic voltage scaling: Myths and facts”. In: *PARTS*. 2005 (cit. on p. 25).

[Sta06] Garrick Staples. “TORQUE Resource Manager”. In: *Supercomputing*. 2006 (cit. on p. 5).

[SV09] Tarry Singh and Pavan Kuman Vara. “Smart Metering the Clouds”. In: *WETICE*. 2009 (cit. on p. 26).

[TC+14] Ghislain Landry Tsafack Chetsa, Georges Da Costa, Laurent Lefevre, et al. “Energy aware approach for HPC systems”. In: *High-Performance Computing on Complex Environments*. 2014 (cit. on p. 57).

[Tsa+05] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. “Modeling user runtime estimates”. In: *JSSPP*. 2005 (cit. on pp. 39, 73, 74).

[Tsa+07] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. “Backfilling using system-generated predictions rather than user runtime estimates”. In: 2007 (cit. on pp. 49, 76, 77, 81, 82, 84, 85, 87).

[Wea06] Patrick Weaver. “A Brief History of Scheduling”. In: (2006) (cit. on p. 5).

[Won+00] Adrian T Wong, Leonid Oliker, William TC Kramer, Teresa L Kaltz, and David H Bailey. “ESP: A system utilization benchmark”. In: *SuperComputing*. 2000 (cit. on pp. 12, 65).

[Yan+13] Xu Yang, Zhou Zhou, Sean Wallace, et al. “Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems”. In: *SuperComputing (SC)*. 2013 (cit. on p. 46).

[Yoo+03] Andy B. Yoo, Morris A. Jette, and Mark Grondona. “SLURM: Simple Linux Utility for Resource Management”. In: *JSSPP*. 2003 (cit. on pp. 5, 32, 56, 58, 61).

Résumé en français

B

B.1 Introduction

Le domaine du calcul haute performance (*i.e.* la science des super-calculateurs) est caractérisé par l'évolution continuelle des architectures de calcul, la prolifération des ressources de calcul et la complexité croissante des problèmes que les utilisateurs veulent résoudre. Un des logiciels les plus importants de la pile logicielle des supercalculateurs est le Système de Gestion des Ressources et des Tâches. Il est le lien entre la charge de travail donnée par les utilisateurs et la plateforme de calcul. Ce type de logiciels spécialisés fournit des fonctions pour construire, soumettre, planifier et surveiller les tâches dans un environnent de calcul complexe et dynamique.

Pour pouvoir atteindre des supercalculateurs exaflopiques, de nouvelles contraintes et objectifs ont été inventés. Cette thèse développe et teste l'idée que les utilisateurs de ces systèmes peuvent aider à atteindre l'échelle exaflopique. Spécifiquement, nous montrons des techniques qui utilisent les comportements des utilisateurs pour améliorer la consommation énergétique et les performances globales des supercalculateurs.

Pour tester ces nouvelles techniques, nous avons besoin de nouveaux outils et méthodes qui sont capables d'aller jusqu'à l'échelle exaflopique. Nous proposons donc des outils qui permettent de tester de nouveaux algorithmes capables de s'exécuter sur ces systèmes. Ces outils sont capables de fonctionner sur de petits supercalculateurs en émulant ou simulant des systèmes plus puissants. Après avoir évalué différentes techniques pour mesurer l'énergie dans les supercalculateurs, nous proposons une nouvelle heuristique, basée sur un algorithme répandu (Easy Backfilling), pour pouvoir contrôler la puissance électrique de ces énormes systèmes. Nous montrons aussi comment, en utilisant la même méthode, contrôler la consommation énergétique pendant une période de temps. Le mécanisme proposé peut limiter la consommation énergétique tout en gardant des performances satisfaisantes. Si l'énergie est une ressource limitée, il faut la partager équitablement. Nous présentons de plus un mécanisme permettant de partager la consommation énergétique entre les utilisateurs. Nous soutenons que cette méthode va motiver les utilisateurs à réduire la consommation énergétique de leurs calculs. Finalement, nous analysons le comportement actuel et passé des utilisateurs pour améliorer les performances des supercalculateurs. Cette approche non seulement surpasse les performances des travaux existants, mais aussi ouvre la voie à l'utilisation de méthodes semblables dans d'autres aspects des Systèmes de Gestion des Ressources et des Tâches.

B.2 Contributions

Les études présentées dans cette thèse on fait l'objet des publications suivantes :

- Conférences internationales avec actes et comité de lecture : [Geo+c], [Geo+b], [Geo+a], [Gau+]
- Conférences internationales:
  - "Road to exascale: what if end-users can help? An approach to respond to new system needs in the batch scheduler", *dans* HPC Days in Lyon 2016, Avril 2016
  - "Improving Backfilling by using Machine Learning to Predict Running Times in SLURM", *dans* Slurm Birds of a Feather (SC15), Novembre 2015
  - "Power Adaptive Scheduling" and "Improving Job Scheduling by using Machine Learning", *dans* Slurm User Group Meeting, Septembre 2015
  - "Adaptive Resource and Job Management for limited power consumption" and "Introducing Energy based fair-share scheduling", *dans* Slurm User Group Meeting, Septembre 2014
- Conférence française:
  - "Ordonnancement dynamique des applications dans les supercalculateurs pour limiter la consommation électrique", *dans* Green Days @Rennes, Juillet 2014

B.3 Outils et méthodologies

B.3.1 Rejeu de traces à large échelle

Le calcul haute performance s'appuie sur une grande variété de logiciels systèmes. Le gestionnaire de ressources et de tâches a une position stratégique dans la pile logiciel puisqu'il a le contrôle de l'infrastructure matériel et de la charge de travail fournis par les utilisateurs. La complexité de ce logiciel ainsi que la pléthore de paramètres et configurations font qu'il est difficile de modéliser son comportement avec des simulateurs.

Cette sous-section introduit une méthodologie expérimentale à large échelle qui permet d'étudier les gestionnaires de ressources et de tâches comme un seul et unique composant. Cette méthodologie est basée sur le rejeu de charge de travail synthétique ou complet, en se concentrant particulièrement sur les conditions d'expérimentation et le post-traitement des résultats.

Elle fournit des techniques génériques qui passent à l'échelle et qui peuvent être utilisées avec différents gestionnaires de ressources et de tâches pour:

- comparer les performances,
- améliorer la configuration,
- évaluer de nouvelles fonctionnalités,
- valider des systèmes réels avant la mise en production et
- prédire le passage à l'échelle de nouvelles algorithmes.

Cette méthodologie est validée avec des expérimentations utilisant deux gestionnaires de ressources et de tâches répandus, à travers divers scénarios en rejouant des traces synthétiques et des sections de la charge de travail du supercalculateur pétaflopique, Curie.

B.3.2 Simunix, un simulateur de plateforme

Pour compléter la méthodologie de la sous-section précédente, nous proposons dans cette sous-section une nouvelle approche pour simuler les gestionnaires de ressources et de tâches. Au lieu de simuler le gestionnaire directement, nous proposons de simuler la plateforme sur laquelle il s'exécute. Cette méthode permet de ne pas avoir à insérer des modifications dans le gestionnaire de ressources et de tâches tout en pouvant simuler diverses plateformes de calcul haut performance.

Pour réaliser ce simulateur de plateforme nous utilisons diverses astuces pérmisent par les systèmes UNIX. La partie simulation s'appuie sur la structure logiciel Simgrid. Simgrid permet de simuler des systèmes distribués et supporte de nombreuses fonctionnalités.

Cette étude en est à ces prémisses et donc aucuns tests ne sont encore présentés.

B.3.3 Acompte et contrôle d'énergie avec le gestionnaire de tâches et de ressources SLURM

La consommation énergétique est devenue graduellement un paramètre important des centres de calcul haute performance. Le gestionnaire de ressources et de tâches est l'intergiciel qui est responsable de distribuer la puissance de calcul aux logiciels et à la connaissance des ressources et du besoin des tâches. Il est donc le meilleur candidat pour surveiller et contrôler la consommation énergétique des calculs par rapport aux spécifications des tâches. L'intégration des mécanismes de mesure d'énergie dans les gestionnaires de ressources et de tâches et la considération de la consommation énergétique comme une nouvelle caractéristique de la comptabilité semblent primordiales en ce moment où l'énergie est devenue le principal frein à l'accroissement des ressources de calculs. Comme les Power-Meters sont trop chers, d'autres modèles de mesures comme IPMI et RAPL peuvent être exploités par le gestionnaire pour pouvoir surveiller la consommation énergétique et améliorer la surveillance des exécutions de tâches.

Dans ce chapitre, nous présentons la méthodologie et l'implémentation d'un nouveau cadre logiciel développé dans le gestionnaire de ressources et de tâches SLURM. Il permet d'effectuer une comptabilité énergétique par tâche, de sauvegarder le profil énergétique temporel et de contrôler la fréquence des processeurs. Comme l'objectif de ce travail est le déploiement de ce cadre logiciel dans un grand centre pétaflopique de calcul comme Curie, son coût et sa fiabilité sont des problèmes importants. Nous évaluons le surcoût de ces choix méthodologiques et la précision des différents modes de mesures en utilisant différentes applications références (Linpack, IMB, Stream) sur une vrai plateforme de calcul. Nos expérimentations montrent que le surcoût est de moins de 0,6% en consommation énergétique et de moins de 0,2% en temps d'exécution.

B.4 Management adaptatif de ressources et de tâches pour une consommation de puissance électrique limitée

Ces dernières décennies ont été caractérisées par un accroissement continuel des ressources de calcul et de stockage. Cette tendance a mis le focus sur l'habilité à gérer efficacement la puissance électrique requise pour opérer les centres de données et de calculs derniers cris. La consommation électrique des supercalculateurs a besoin d'être ajustée par rapport à un budget de puissance électrique évolutif ou à la disponibilité de l'électricité. Les gestionnaires de ressources et de tâches ont donc dû s'adapter pour ordonnancer les tâches avec des performances optimisées tout en limitant la puissance électrique consommée.

Ce chapitre introduit une nouvelle stratégie d'ordonnancement qui fournit la capacité d'adapter, de façon autonome, la charge de travail à une puissance électrique donnée. L'originalité de cette approche repose sur une combinaison du DVFS (Dynamic Voltage and Frequency Scaling ou ajustement dynamique du voltage et de la fréquence) et de l'extinction des nœuds de calculs pour contrôler la puissance électrique. Elle a été implémentée dans un gestionnaire de ressources et de tâches célèbre, SLURM. Finalement, nous avons validé cette approche grâce à des émulations de plateformes larges échelles en utilisant la charge de travail d'une plateforme pétaflopique, Curie.

B.5 Contrôle de budget grâce à un ordonnanceur prenant en compte la consommation énergétique

La consommation énergétique est devenu une des issues critiques pour l'évolution des centres de calculs haute performance. Contrôler la consommation énergétique de ces plateformes n'est pas seulement un moyen de contrôler le coût de celle-ci mais c'est aussi une étape nécessaire pour attendre les plateformes exaflopiques. Les

décisions du gestionnaire de ressources et de tâches en matière d'ordonnancement et de gestion de ressources ont un impact direct sur la consommation énergétique totale. Le Powercapping (la limitation en puissance électrique) est une technique grandement étudiée qui garantit que la plateforme ne dépassera pas un seuil de puissance électrique. En revanche, elle manque de flexibilité pour ordonnancer efficacement les jobs dans la durée.

Nous proposons un mécanisme d'ordonnancement de tâches qui étend l'algorithme de Backfilling pour prendre en compte la consommation énergétique tout en adaptant la gestion des ressources grâce à une technique d'extinction des nœuds de calculs. Cette combinaison permet un contrôle efficace de la consommation énergétique pour respecter un budget donné pendant une période de temps. Cette technique est expérimentée, validée et comparée avec plusieurs alternatives à travers un grand nombre de simulations. Ces expériences montrent une haute utilisation système ainsi que des conséquences intéressantes pour l'efficacité énergétique dans ces plateformes.

B.6 Un mécanisme incitatif au niveau de l'ordonnanceur pour améliorer l'efficacité énergétique des supercalculateurs

Alors qu'il existe un grand nombre d'algorithmes et de techniques de programmation pour économiser de l'énergie, les utilisateurs n'ont pas de motivations à les utiliser puisque cela pourrait réduire les performances de leurs tâches.

Nous proposons de gérer dans ce chapitre le budget des supercalculateurs grâce à EnergyFairShare (EFS), un algorithme d'ordonnancement ressemblant à FairShare. FairShare est une règle d'ordonnancement classique qui donne plus de priorité aux tâches appartenant à des utilisateurs qui ont consommé peu de temps CPU dans le passé. De la même manière, EFS va suivre la consommation énergétique des utilisateurs et prioriser les tâches de ceux qui consomment le moins. Ainsi, EFS va motiver les utilisateurs à optimiser leurs calculs pour une plus grande efficacité énergétique. Avec une plus grande priorité, les tâches vont moins attendre pour s'exécuter et ainsi les utilisateurs auront leurs résultats plus rapidement.

Pour valider ce principe, nous avons implémenté EFS dans un simulateur et utilisé des charges de travail de plusieurs centres de calculs. Les résultats montrent qu'en diminuant sa consommation énergétique un utilisateur va réduire son temps d'attente pour obtenir ses résultats. Pour valider la faisabilité de notre approche, nous avons aussi implémenté EFS dans SLURM, un gestionnaire de ressources et de tâches répandu.

Nous avons validé notre extension à SLURM en émulant une plateforme large échelle et en expérimentant sur une vraie plateforme. Nous observons encore une

fois que les utilisateurs avec une meilleure efficacité énergétique attendent moins longtemps leurs résultats.

B.7 Amélioration du Backfilling grâce à l'utilisation d'algorithmes d'apprentissage automatique pour apprendre le temps d'exécution des tâches

Tandis que les centres de calculs génèrent toujours plus de données, ils sont caractérisés par des incertitudes sur des paramètres comme le temps d'exécution de logiciels. La question soulevée par ce travail est : est-il possible et utile de se servir des prédictions du temps d'exécution des logiciels pour améliorer l'ordonnancement global ?

Nous présentons une étude complète pour répondre à cette question en faisant l'hypothèse que la politique populaire EASY Backfilling est utilisée. Plus précisément, nous nous appuierons sur des méthodes d'apprentissage automatique classiques et nous proposeront de nouvelles fonctions de coûts adaptés au problème. Ensuite nous testerons nos solutions à travers de nombreuses simulations utilisant de vraies charges de travail. Enfin, nous proposerons un nouvel algorithme qui surpasse la politique EASY Backfilling de 28% si on considère la mesure de Ralentissent Délimité (Bounded Slowdown).

B.8 Conclusion

Les gestionnaires de ressources et de tâches sont des composants essentiels pour la communauté du calcul haute performance afin de pouvoir atteindre des systèmes de taille exaflopique. Les systèmes de cette taille ouvriront la voix aux chercheurs à des calculs plus gros ainsi que des simulations plus précises. La consommation énergétique et les performances d'ordonnancement sont des points clés de ces futurs centres. A travers différentes études, nous avons montré que nous pouvons contrôler efficacement la puissance électrique et la consommation énergétique, ainsi qu'utiliser le comportement des utilisateurs pour améliorer les performances d'ordonnancement.

Abstract

The field of High Performance Computing (HPC) is characterized by the continuous evolution of computing architectures, the proliferation of computing resources and the increasing complexity of applications users wish to solve. One of the most important software of the HPC stack is the Resource and Job Management System (RJMS) which stands between the user workloads and the platform, the applications and the resources. This specialized software provides functions for building, submitting, scheduling and monitoring jobs in a dynamic and complex computing environment.

In order to reach exaflops HPC systems, new constraints and objectives have been introduced. This thesis develops and tests the idea that the users of such systems can help reaching the exaflopic scale. Specifically, we show and introduce new techniques that employ users behaviors to improve energy consumption and overall cluster performances.

To test the proposed techniques, we need to develop new tools and methodologies that scale up to large HPC clusters. Thus, we designed adequate tools that assess new RJMS scheduling algorithms of such large systems. These tools are able to run on small clusters by emulating or simulating bigger platforms. After evaluating different techniques to measure the energy consumption of HPC clusters, we propose a new heuristic, based on the popular Easy Backfilling algorithm, in order to control the power consumption of such huge systems. We also demonstrate, using the same idea, how to control the energy consumption during a time period. The proposed mechanism is able to limit the energy consumption while keeping satisfying performances. If energy is a limited resource, it has to be shared fairly. We also present a mechanism which shares energy consumption among users. We argue that sharing fairly the energy among users should motivate them to reduce the energy consumption of their applications. Finally, we analyze past and present behaviors of users using learning algorithms in order to improve the performances of the parallel platforms. This approach does not only outperform state of the art methods, it also shows promising insight on how such method can improve other aspects of RJMS.

Résumé

Le domaine du calcul haute performance (*i.e.* la science des supercalculateurs) est caractérisé par l'évolution continuelle des architectures de calcul, la prolifération des ressources de calcul et la complexité croissante des problèmes que les utilisateurs veulent résoudre. Un des logiciels les plus importants de la pile logicielle des supercalculateurs est le Système de Gestion des Ressources et des Tâches. Il est le lien entre la charge de travail donnée par les utilisateurs et la plateforme de calcul. Ce type de logiciels spécialisés fournit des fonctions pour construire, soumettre, planifier et surveiller les tâches de calculs dans un environnent complexe et dynamique.

Pour pouvoir atteindre des supercalculateurs exaflopiques, de nouvelles contraintes et objectifs ont été inventés. Cette thèse développe et teste l'idée que les utilisateurs de ces systèmes peuvent aider à atteindre l'échelle exaflopique. Spécifiquement, nous montrons des techniques qui utilisent les comportements des utilisateurs pour améliorer la consommation énergétique et les performances globales des supercalculateurs.

Pour tester ces nouvelles techniques, nous avons besoin de nouveaux outils et méthodes capables d'aller jusqu'à l'échelle exaflopique. Nous proposons donc des outils qui permettent de tester de nouveaux algorithmes capables de s'exécuter sur ces systèmes. Ces outils sont capables de fonctionner sur de petits supercalculateurs en émulant ou simulant des systèmes plus puissants. Après avoir évalué différentes techniques pour mesurer l'énergie dans les supercalculateurs, nous proposons une nouvelle heuristique, basée sur un algorithme répandu (Easy Backfilling), pour pouvoir contrôler la puissance électrique de ces énormes systèmes. Nous montrons aussi comment, en utilisant une méthode semblable, contrôler la consommation énergétique pendant une période de temps. Le mécanisme proposé peut limiter la consommation énergétique tout en gardant des performances satisfaisantes. Si l'énergie est une ressource limitée, il faut la partager équitablement. Nous présentons de plus un mécanisme permettant de partager la consommation énergétique entre les utilisateurs. Nous soutenons que cette méthode va motiver les utilisateurs à réduire la consommation énergétique de leurs calculs. Finalement, grâce à un algorithme d'apprentissage automatique, nous analysons le comportement actuel et passé des utilisateurs pour améliorer les performances des supercalculateurs. Cette approche non seulement surpasse les performances des travaux existants, mais aussi ouvre la voie à l'utilisation de méthodes semblables dans d'autres aspects des Systèmes de Gestion des Ressources et des Tâches.