Towards Energy Budget Control in HPC

Pierre-François Dutot*, Yiannis Georgiou†, David Glesser†, Laurent Lefevre‡, Millian Poquet* and Issam Rais‡
*Univ. Grenoble Alpes
surname.name@imag.fr
†Bull Atos Technologies
surname.name@atos.net
‡Univ. Lyon, Inria, CNRS, ENS de Lyon, Univ. Claude-Bernard Lyon 1, LIP
surname.name@inria.fr

***Abstract*—Energy consumption has become one of the most critical issues in the evolution of High Performance Computing systems (HPC). Controlling the energy consumption of HPC platforms is not only a way to control the cost but also a step forward on the road towards exaflops. Powercapping is a widely studied technique that guarantees that the platform will not exceed a certain power threshold instantaneously but it gives no flexibility to adapt job scheduling to a longer term energy budget control.**

We propose a job scheduling mechanism that extends the backfilling algorithm to become energy-aware. Simultaneously, we adapt resource management with a node shutdown technique to minimize energy consumption whenever needed. This combination enables an efficient energy consumption budget control on a cluster during a period of time. The technique is experimented, validated and compared with various alternatives through extensive simulations. Experimentation results show high system utilization and limited bounded slowdown along with interesting outcomes in energy efficiency while respecting an energy budget during a particular time period.

***Index Terms*—HPC; Resource Management; Scheduling; Energy Budget;**

I. Introduction

Modern supercomputers run on huge amounts of electrical power. For instance, Sunway TaihuLight (the leading system of the TOP500) develops 93 petaflop/s and consumes a power of 15 megawatts (MW)[1]. For such platforms, the electricity bill for their lifespan can be roughly equal to their hardware cost. The energy consumption is the most important obstruction for building exascale machines [1].

To control the energy consumption of such huge platforms, multiple techniques have been developped. One of them, powercapping, limits the power consumption to a certain threshold. Limiting this during a time period leads to controlling the *energy* consumption (as $\int power.dt = energy$). The survey provided in [2] gives a thorough analysis of related work on power management strategies along with details upon the relationship between supercomputing centers and electricity service providers in the US. Among the different studied techniques, powercapping in conjuction with job scheduling and shutdown mechanisms appeared as the most promising. By employing these methods we can manage energy consumption through the control of the instant power consumption. Nevertheless, they lack in adaptability, given that power is controlled independently of the instant load. A recent study [3], implicating a larger group of supercomputers and electricity providers in both the US and Europe, showed that while the upper power-bound is an important parameter, power variations do not affect the final energy cost in most use cases. In this paper, we show that adopting flexible power adaptive scheduling techniques, by setting restriction on energy consumption instead of power, can optimize system utilization, slowdown and even energy efficiency when compared to rigid powercapping strategies.

We present an adaptation of a standard job scheduling algorithm that is able to limit the energy consumption during a time period. It is similar to powercapping, with the difference that instead of limiting execution under a maximum instantaneous power consumption, we limit the maximum energy consumption for a particular time duration. The developed techniques are extensions to the backfilling mechanism [4]. Instead of considering only the availability of computing resources, they also take the availability of energy into account. Overall, they enable the platform to meet a certain energy consumption budget. The reduction of energy consumption takes place through idling or opportunistic shutdown of nodes. Experimentations through intensive simulations show that our techniques keep high performances while respecting specific energy budget objectives.

We start by describing the problem in Section II. Then, we present and discuss in Section III the main existing approaches to control power and energy. Section IV presents our new algorithms to support energy budgeting. Then, Section V reports the simulations done with actual log data. Finally, in Section VI, conclusions and future works are presented.

II. Problem Description

A. Scheduling Jobs in HPC

In our context, a job is an application that a user wants to execute on a computing platform. In order to execute those jobs, the scheduler needs to determine when and where they should be executed, based on some algorithm. Usually, only limited information is known about the jobs in practice. Indeed, users typically specify the number of computational resources they need and an execution time upper bound, which is called the walltime – if a job reaches its walltime, it is killed. Sometimes a few more constraints can be specified,

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

such as the minimum amount of memory required. In this work, we define a job j by its arrival date r_j (called *release date* in the scheduling community), its number of requested processors m_j, an estimation of the running time $wall_j$ and the real running time p_j (p_j is not known in advance and is called *processing time* in the scheduling community).

We do not consider the network hierarchy existing in most HPC centers. More simply, we consider that all processors are totally ordered, and that a job must run on neighboring processors. This simplification reduces the complexity of allocating jobs while being close (or even equal) to actual scheduling problems.

B. Energy and Power-saving Techniques

We use a simplistic model of energy consumption in which a processor may have three states: In case it is idle, it consumes P_{idle}; If a job is running on it, it consumes P_{comp}; And if it is switched off, it consumes P_{off}.

In order to reduce and control the energy consumption, there exist techniques which allow energy savings on different levels. Some of these techniques are introduced by architecture manufacturers (e.g. DVFS), others are invariant possibilities of the infrastructure (e.g. node On/Off, heterogeneity, etc). In this work, we take into consideration only the shutdown technique and we consider that each processor can be switched off independently.

The shut down of a processor consumes $P_{on\to off}$, during $t_{on\to off}$. The power on of a processor consumes $P_{off\to on}$, during $t_{off\to on}$.

Despite the simplicity of this model, we think that it meets our needs for two reasons. First, the experiments that we conducted (see section V-B) show that it is sufficient and summarizes well our use cases. Second, a more precise measure of energy would have a prohibitive cost: a hardware cost (integrated energy sensors are not accurate enough [5]), a software architecture cost (dedicated software for accurate energy measurements should be added over the cluster), and a data management cost (to store and analyze the data). A basic energy measurement reduces the cost of all these steps.

Readers may wonder why we do not take DVFS into account, since DVFS can be used to control the power consumption of a job. Previous studies [6], [7] have shown that controlling the energy consumption of jobs with DVFS is not trivial. Depending on the type of application, a given DVFS value may either increase or decrease the total energy consumption of the application. Thus, without a precise knowledge about each job, the scheduler cannot guarantee that a given DVFS value will decrease the energy consumption. Since the scheduler usually does not have this type of information, we think that dynamically adapting the DVFS to reduce the energy consumption should be done within the job itself, not at the scheduling level, as it will result in better energy efficiency.

Similar to DVFS another technique to dynamically adjust the power consumption of sockets is RAPL (Running Average Power Limit) [8], [9]. This technique is particular to modern Intel processors and our goal in this article is to have a solution that would be adapted to any kind of architecture including ARM and AMD.

Nevertheless, we would like to outline that our approach is totally compatible with job-level DVFS optimisations or RAPL powercapping.

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

C. Scheduling Algorithms Evaluation

No perfect scheduling objective exists [10]. However, in this paper, we will consider three different measures. The first measure is the utilization : The proportion of processors that have been used during a time period. This objective is mostly used by cluster owners, as it may represent the productivity of a cluster.

The bounded slowdown [11] (or bounded stretch) is more end-user centered. It tries to measure how fast an end-user will obtain its result. Most of the time its average is computed:

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

where $wait_j = start_j - r_j$ is the waiting time of job j ($start_j$ is the moment at which job j starts to be executed), τ is the bounding constant (generally set to 10 seconds in the literature), and n the number of jobs. The third measure is of course the energy consumed. In this paper we do not try to reduce it, but to control it in a period of time.

III. Related Work

A. Controlling Power and Energy Consumption

Many papers focus on controlling the power consumption [12], [9], [13]. In these studies, the objective is to control the final energy cost of the cluster while keeping good performances. Patki et al. [14] 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: they may require high knowledge about the running applications (to tune DVFS or a similar technique), and it also delays some jobs. In our previous study [6], 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, as we want to keep the benefit of controlling the cost of the cluster, while not discriminating against any type of job.

Gholkar et al. in [15] presented a 2-level hierarchical powercapping solution based on RAPL technique which is certainly more adapted than DVFS to guarantee a global powercap.

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

A quite similar energy budget policy has been studied in [17]. The algorithm was implemented upon LSF which is a proprietary resource and job management system and it makes use of CPU Frequency scaling technique. The authors claim that while their policy manages to control the cluster's energy budget they did not observe any energy reduction. In contrast to that work our energy budget mechanism studied in this article is based on a processor shut-down technique and under particular conditions we did observe improvements in energy efficiency.

In [18], Murali et al. study a metascheduler that controls multiple HPC centers. The objective is to reduce the overall cost by adapting the energy consumption to the electricity price of each different cluster. Yang et al. [19] consider the scheduling problem with 2 periods: one during which an energy limit is set, and the other one without energy limit. While this approach is interesting, the algorithm they used is not scalable and is hardly extendable with other constraints. In the study [20], Khemka et al. maximize a "utility" function in a cluster with daily energy budget. They solve the problem thanks to an offline heuristic. Instead of relying on an utility function, we use classic scheduling objectives as described in Section II-C.

The energy consumption of a shut-down node is very small [21]. The technique called opportunistic shutdown takes advantage of this power saving. This technique consists of shutting down the nodes that are idle. To do so, the nodes are monitored. As soon as a defined idle period is witnessed, the decision of shutting them down is made. As shown in [22], such a solution could lead to non-negligible energy savings. However, this solution has some limitations. One of them is the cost, in both time and energy, which is involved by switching nodes on or off. Going off and on again can take several minutes at maximum power [23].

B. Resource and Job Management Systems

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

Managing resources at this scale compels the scheduling algorithm to be very efficient. Therefore, greedy algorithms such as EASY Backfilling [4] are commonly used in HPC centers. Unfortunately, these algorithms do not take energy consumption into account during their decision process.

The EASY Backfilling algorithm – summarized in Algorithm 1 – is one of the most widely used scheduling algorithms in the systems we are interested in. This algorithm only focuses on the present time since the future is unpredictable as we do not know beforehand several events, like node failures or the ending of jobs (the running time p_j is unknown before it happens). 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 ease of extending the basic policy, 3) the high resource utilization rate implied by this aggressive backfilling policy, and 4) the scalability of being present-focused.

Algorithm 1: The EASY Backfilling algorithm

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

IV. Proposed Algorithm

A. Desired Properties

Obviously, the proposed algorithm should comply with an energy budget during a given period. This energy budget should be strictly respected. Moreover, we wanted the algorithm to be modular enough to support extra features, like opportunistic shutdown. Since large-scale platforms are targeted by our algorithm, we wanted it to be efficient on them. Finally, for the purpose of maximizing our algorithm adoption, we wanted it to avoid dramatic changes over currently implemented solutions.

B. Algorithm Description

As EASY Backfilling (summarized in Algorithm 1) has all the desired properties but the energy ones, we chose to base our algorithm on it. In order to comply with the energy budget, we defined two rules which our algorithm must respect under all circumstances.

- Rule 1: Avoid spending the whole budget too early, as it would unbalance the performances during the budget

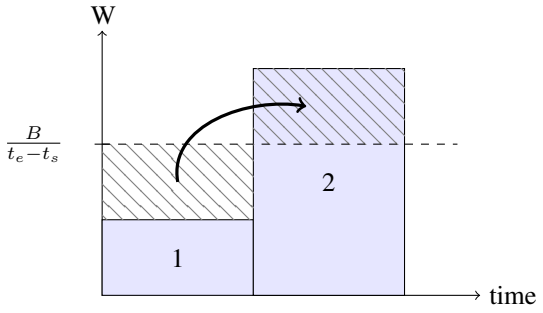


Fig. 1: Main idea behind our algorithms: set a power limit then use the saved energy to start jobs that exceed the power limit. Here, the energy saved during job 1 is then used to run job 2.

period. To this end, the budget's energy is made available gradually over time, at a rate of $B/(t_e - t_s)$ joules per second.

- Rule 2: Never have energy debts. Thus, before taking the decision of running a job, we have to ensure that enough energy is available for the entire duration of the job execution. This comprises taking the past, present and future power consumption of the whole cluster into account.

If we replace *energy* by *money* and *running jobs* by *buying stuff*, these two rules could describe how someone that never wants to be in debt would manage its monthly paycheck. Figure 1 shows the global idea behind these two rules: we save energy by consuming less than the power rate and, then, we can use this energy to run jobs that require more energy than the power rate to run.

These 2 rules are integrated within the EASY Backfilling algorithm in the following manner. A counter named C_{ea} stores the amount of available energy, *i.e.* the amount of energy that the algorithm is allowed to spend at the present time. C_{ea} equals to the amount of energy made available since the beginning of the budget period (via rule 1), minus the energy which has been consumed by the cluster. Whenever the algorithm checks whether enough processors are available to run a job, it also checks whether enough energy is available. The energy balance should always be positive during the entire execution time of the job (to fulfill rule 2). The algorithm uses $wall_j$ to estimate the length of job j.

Whenever the job at the top of the queue cannot be started immediately, a processor reservation is made for it (as in regular EASY Backfilling). The backfilling rule dictates that other jobs might be executed before the first job if they do not delay it. However, because of our set of rules, if jobs were backfilled the usual fashion, they might delay the first job by *stealing* its energy. Consequently, our algorithm also makes energy reservation for the first job when it cannot be started immediately.

Algorithm 2 describes our algorithm. The additions we have done over EASY Backfilling to support energy budgets are underlined and colored in brown.

Algorithm 2: Energy Budget Backfilling Algorithm

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

C. Implementation details

1) Energy Consumption Monitoring: The counter C_{ea}, which stores the amount of available energy, is updated whenever the algorithm is called but also at every *monitoring stage*.

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

Monitoring stages allow to obtain the amount of energy which has really been consumed by the cluster. Therefore, C_{ea} is incremented during these stages (because the algorithm always overestimates this amount of energy). The stages occur periodically every *monitoring stage*. The period value must be set considering a trade-off between precision and the overhead of gathering energetic data. The more precise the monitoring of the energy is, the more precisely our algorithm will respect the energy budget. Increasing the monitoring period increases overheads as a fraction of the computing resources has to be used to gather and transmit the data.

2) Energy Consumption Estimation: In the above explanation of the algorithm, it is explained that the algorithm must determine whether the energy balance would be positive in the

future. As accurately predicting the energy consumption of a job (or a cluster) is difficult, we wanted our algorithm to work even when estimations are imprecise.

We assume that, to predict the energy consumption of the jobs (and of the cluster), using overestimated values is enough to make our algorithm work.

We note $\tilde{P}_{comp}$ the estimation of the power consumption of one computing processor (in watts), and $\tilde{P}_{idle}$ the estimation of the power consumption of one idle processor (in watts). With overestimated $\tilde{P}_{comp}$ and $\tilde{P}_{idle}$, the algorithm will overestimate the energy consumption of the cluster in the future. However, at every monitoring stage, the real energy consumption is measured and C_{ea} is updated with the available energy minus the energy actually consumed. Thus, this overestimation leads to saving more energy in the counter, which will allow more jobs to be started later on.

In order to obtain such overestimations, we recommend to benchmark the power consumption of the cluster in certain scenarios and to use the maximum observed value. To estimate $\tilde{P}_{comp}$, executing a CPU-intensive application (e.g. one from the LINPACK suite) is recommended. To estimate $\tilde{P}_{idle}$, observing processors doing nothing is recommended.

3) Interactions with opportunistic shutdown: The algorithm we proposed does not need to be modified to work with opportunistic shutdown. Indeed, the power consumption of an off node is lesser than an idle node's one. This leads our algorithm to overestimate even more the cluster's energy consumption, which would make a greater amount of energy available after monitoring stages.

4) Differences with EASY Backfilling: The EASY Backfilling algorithm is called whenever a job arrives or some resources are freed. Our algorithm is run in the same cases, but also when more energy is available, namely at every $monitoring\ stage$. The overhead is minimal, as the full algorithm is only run if the amount of available energy is sufficient to run the first job of the queue.

In the particular case of the energy budget is unlimited ($B = \infty$), our algorithm produces the same schedules as EASY Backfilling's ones. Additionally, if the energy budget B is very small, our algorithm will start the jobs in the order of the queue.

D. An alternative similar to powercap

The proposed algorithm is close to a powercap mechanism. Making $B/(t_e - t_s)$ joules available each second is close to having a powercap limit of $B/(t_e - t_s)$. The rules introduced previously can be seen as rules which allows to violate the powercap in some cases (these cases being mostly "when energy is available").

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

The difference between $energyBud$ and $reducePC$ lies in how the jobs respect their energy reservation. In $energyBud$, the reserved energy is substracted from C_{ea}. In $reducePC$, we reduce the number of joules made available per second (as if the powercap had been reduced). If job i makes a reservation of J_i joules at time q_i (and thus guarantees to start at q_i), the number of joules available per second is reduced by $J_i/(q_i - now)$ during the time period between now and q_i.

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

V. Evaluation

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

- How better are we compared to a standard powercap mechanism?
- What is the gain of activating opportunistic shutdown?
- If the budget is reduced by 80%, are the performances also reduced by 80%?
- Which is the best one, $reducePC$ or $energyBud$?

The scripts used to produce, analyze and visualize the results are available in a git repository to allow other scientists to reproduce and improve them[2].

A. Simulator

In order to evaluate our algorithms, we chose to analyze their behavior through simulations. For this purpose, we used Batsim [24], a scheduler simulator based on SimGrid [25]. 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.

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

B. Simulation Calibration

To calibrate our simulator, we made various measurements on the Taurus Grid5000 [26] cluster. This cluster is composed of 16 Dell PowerEdge R720 nodes, each with two Intel Xeon E5-2630. The nodes are equipped with wattmeters, allowing to measure precisely their energy consumption (one value every second).

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

[2]https://github.com/glesserd/energybudget-expe

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

TABLE I: The values used to calibrate the simulator and to parameterize our algorithms.

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

In order to obtain switch-related measurements, we made 50 "switch on" and 50 "switch off" operations on each node. We consider a node to be off when the consumed power reaches its minimum and on when the node is capable of starting a new job (*i.e.* when all services are running and operational). We were able to measure the amount of time and the amount of consumed energy of each switch operation. We then chose to average these amounts to obtain $P_{off \to on}$, $t_{off \to on}$, $P_{on \to off}$ and $t_{on \to off}$.

Finally, we chose a $monitoring\ period$ of 10 minutes, as it appears to be a good trade-off between precision and monitoring overhead. This choice is complex as it depends on the available energy sensors and the way to gather data from the computing nodes to the controlling node. Simulations have to be close enough to the reality and thus we choose a value close to what seems to be used in various supercomputers.

C. Testset

To assess our algorithms, we chose to replay 1-week-long extracts of real traces, available on the Parallel Workload Archive [27]. We chose to use 3 different traces and to extract 10 disjoint weeks from each one of them, thus leading to 30 different input workloads for our simulator. Since scheduling decisions have more impact when the utilization is high, the weeks have been selected with this criterion in mind. The original traces are:

- Curie (80640 processors, dates from 2012 and lasts 3 months),
- MetaCentrum (3356 processors, dates from 2013 and lasts 6 months),
- SDSC-Blue (1152 processors, dates from 2003 and lasts 32 months).

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

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

Furthermore, each algorithm which supports power budgeting is executed with and without opportunistic shutdown. When the opportunistic shutdown mechanism is enabled, as soon as idle nodes are witnessed , the decision of shutting them down is made. Finally, our evaluation process comprises 1470 configurations, which have all been simulated.

D. Results

Each trace does not come from the same cluster and does not have the same jobs. Thus, they do not present the same values for the measures, nor the same opportunities for the algorithms to improve results. As a consequence, to be able to compare results, all measures are normalized to reduce the effect of each trace. The method described in [28] normalizes the data to remove the between-subject variability. All of the following graphs present means of the normalized measures considering the traces as the between-subject variable.

Also, as stated in sub-section II-C, some measures are defined for a time period and some for a number of jobs. For the ones that depend on a time period (utilization and relative energy consumed), we use the whole week as the time period. Thus, the jobs that have been scheduled after the end of the week are not taken into account in these measures. For the ones that depend on a number of jobs (AVEbsld and number of job started), all jobs of each trace are taken into account.

E. How better are we compared to a standard powercap mechanism?

Figure 2 depicts the normalized mean utilization for each experimental condition and energy budget. The black line is explained in Section V-G. We observe that $energyBud$ outperforms the other algorithms. $reducePC$ performs better than $energyBud$ when opportunistic shutdown is activated for

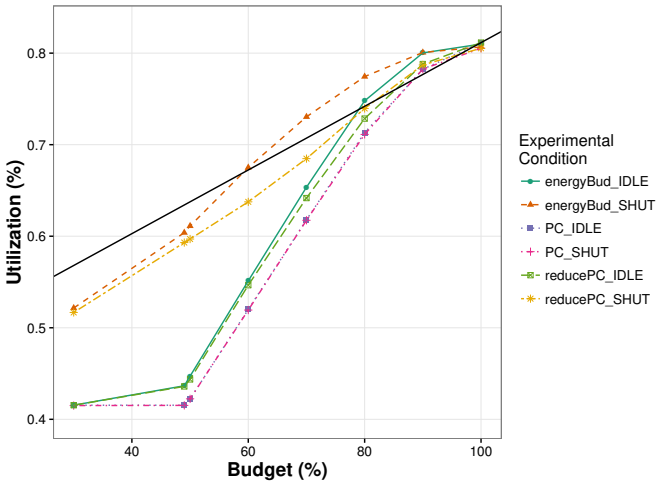


Fig. 2: Normalized mean system utilization during the week. The black line represents a theoretical performance baseline.

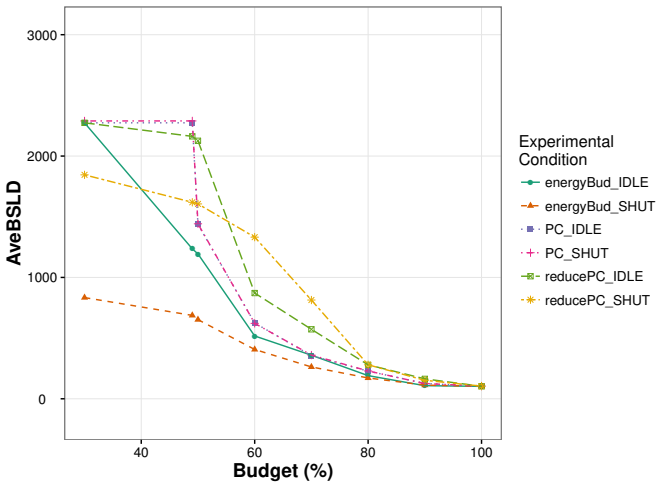


Fig. 3: Normalized mean AVEbsld for all jobs.

the former and deactivated for the latter. As expected, below an energy budget of 49%, all experiments without opportunistic shutdown have the same results – PC_SHUT also has the same performance as the PC mechanism cannot benefit from the energy saved thanks to the opportunistic shutdown.

In Figure 3, the normalized mean AVEbsld for each experimental condition and energy budget is shown. The AVEbsld are very high because we choose traces with a high utilization. In a production system, a high utilization also means a lot of jobs waiting in the queue. AVEbslds increase even more for low energy budgets as the used resources are limited during a fair part of the week. Once again, $energyBud$ with and without the opportunistic shutdown outperforms all other algorithms. Surprisingly, the $powercap$ mechanism is not the worst. The same results are obtained with and without energetic shutdown as expected for an algorithm that does not benefit from a reduced energy consumption.

In terms of energy, Figure 4 presents the normalized mean energy consumed during the week relative to the total energy

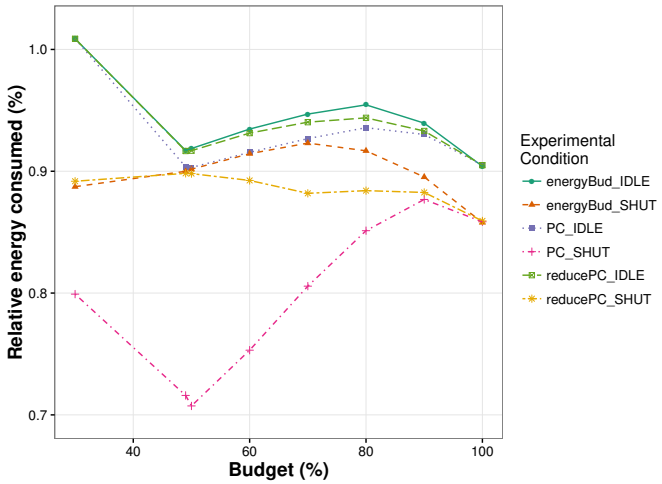


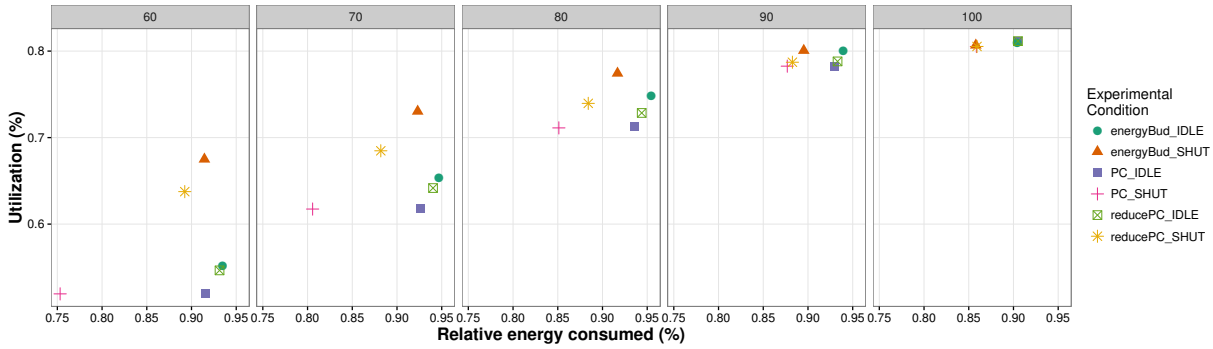
Fig. 4: Normalized mean energy consumed during the week relative to the maximum energy consumable during the same period.

consumable during the same period. Algorithms without opportunistic shutdown consume more than the energy available when the budget is set at 30%, because at 30% the energy budget is below the energy consumption of the cluster fully idle. 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 $powercap$. Our algorithms do not try to minimize the energy consumption. They try to keep it under a certain value. This behavior can be observed in 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 has a high utilization while having a high energy consumption.

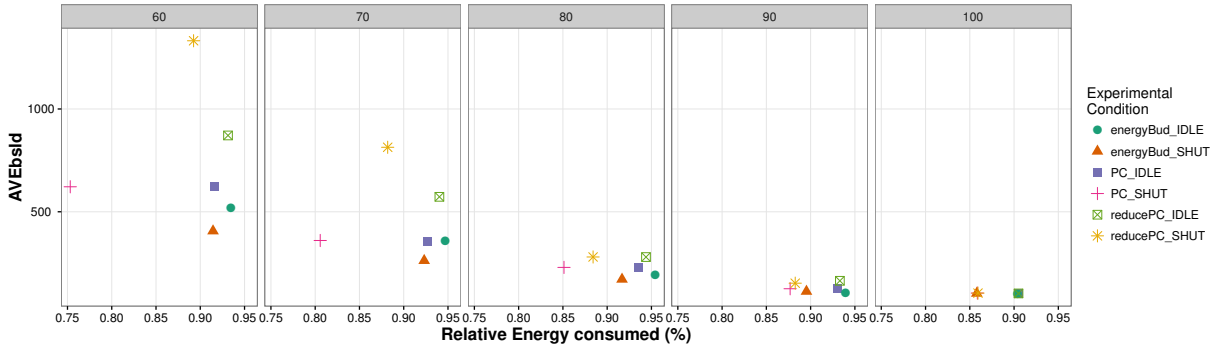
F. What do we gain by employing opportunistic shutdown?

Table II shows the average performance difference of different measures when the opportunistic shutdown is employed. To compute this table, we take every experimental condition on each trace with opportunistic shutdown and compared it to the same experiment without opportunistic shutdown. The comparaison is done by computing the percent change of opportunistic shutdown over idle: $(y_{SHUT} - y_{IDLE})/y_{IDLE}$, where y_{SHUT} is a normalized measure with opportunistic shutdown activated and y_{IDLE} the normalized measure in the exact same experimental setting (same algorithm and same budget) as y_{SHUT} but without opportunistic shutdown activated. This table presents the average of these computations.

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



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



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

Fig. 5: Performance/energy trade-offs for each algorithm at different energy budgets.

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

TABLE II: Average improvements on different measures when activating shutdown. For AVEbsld and Energy, negative values are better

the activation of opportunistic shutdown dramatically reduces the AVEbsld of $energyBud$.

G. If the budget is reduced by 80%, are the performances also reduced by 80%?

In figure 2, the black line represents a theoretical performance baseline. If we reduce the energy budget by a certain amount, one can expect the performance to decrease by the same amount. This is what this line represents. The line is not the identity because the energy budget only lasts 3 days during the 7 days considered. Thus, this theoretical performance baseline is formulated as:

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

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

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. Presumably, the small limitation in energy reduces the fragmentation and thus improves the utilization. For $energyBud$ with opportunistic shutdown, this is also true for energy budgets of 60% and above. Without opportunistic shutdown, this is only true for 80% and above.

If we go further in this analysis, we can take a look at which algorithm has the best energy/performance trade-off. Figure 5a shows the normalized mean utilization versus the normalized relative mean energy consumption for each experimental condition at different energy budget. The best points in term of energy-performance trade-off are the most upper left ones. The difference when opportunistic shutdown is activated and when it is not can be clearly seen here: when it is activated the points are in the upper left part of the graphs. $energyBud$ with opportunistic shutdown has the best trade-off. $powercap$ is on the Pareto curve but with a very low utilization.

On Figure 5b, a trade-off between the normalized mean AVEbsld and the normalized relative energy consumed is shown. Here, it is the best points that are the lower left ones. Again, $powercap$ with opportunistic shutdown has a

good trade-off because of its low energy consumption. However, $energyBud$ with opportunistic shutdown has the best AVEbsld trade-off. If we only look at the points without opportunistic shutdown, $powercap$ and $energyBud$ have also the best energy/AVEbsld trade-offs.

H. Which is the best one, reducePC or energyBud?

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

VI. Conclusion

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

We proposed two new alternatives and showed their effectiveness on a wide range of scenarios, which have been assessed through simulation. These two new algorithms not only provide a way to control the energy consumption of computing platforms, but they also optimized metrics such as system utilization and bounded slowdown.

Moreover, when the amount of available energy is large, the algorithms we proposed improve the energy efficiency of the cluster.

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

As future work, we will implement our algorithm upon a real open-source resource and job management system such as Slurm or OAR and study its effects in a supercomputer in production. Another extension we are going to consider for energy reductions is to make use of techniques such as CPU powercapping based on RAPL [8]. Furthermore, we will evaluate the integration with job-level runtime systems such as GEO [29], currently under development, whose goal is to dynamically adapt frequency and power allocations across nodes to improve the jobs' power efficiency.

One limitation of our approach is that we only considered periods with a fixed energy budget. Hence, this work could be extended to become more dynamic: If the energy budget followed electricity price, we could control and thus reduce a significant part of the cluster's costs. This would provide an improved solution to the remaining use cases described in [3] whom electricity cost vary since it may partially depend on renewable sources.

Acknowledgment

The work is partially supported by the ANR project MOEBUS. This work is also integrated and supported by the ELCI project, a French FSN ("Fond pour la Société Numérique") project that associates academic and industrial partners to design and provide software environment for very high performance computing. We thank gracefully all the contributors of the Parallel Workloads Archive for making the workloads available.

References

[1] J. Dongarra *et al.*, "The international exascale software project roadmap," in *International Journal of High Performance Computing Applications*, 2011.

[2] N. J. Bates *et al.*, "Electrical grid and supercomputing centers: An investigative analysis of emerging opportunities and challenges," *Informatik Spektrum*, 2015.

[3] T. Patki *et al.*, "Supercomputing centers and electricity service providers: A geographically distributed perspective on demand management in europe and the united states," in *International Supercomputing Conference (ISC)*, 2016.

[4] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *Transactions on Parallel and Distributed Systems (TPDS)*, 2001.

[5] Y. Georgiou, T. Cadeau, D. Glesser, D. Auble, M. Jette, and M. Hautreux, "Energy Accounting and Control with SLURM Resource and Job Management System," in *International Conference on Distributed Computing and Networking (ICDCN)*, 2014.

[6] Y. Georgiou, D. Glesser, and D. Trystram, "Adaptive Resource and Job Management for Limited Power Consumption," in *International Parallel and Distributed Processing Symposium Workshop (IPDPS) Workshop*, 2015.

[7] 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.

[8] Intel, "Intel 64 and ia-32 architectures software developer's manual, volume 3b. system programming guide, part 2." http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf.

[9] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz, "Beyond DVFS: A first look at performance under a hardware-enforced power bound," in *Parallel and Distributed Processing Symposium (IPDPS) Workshops*, 2012.

[10] E. Frachtenberg and D. G. Feitelson, "Pitfalls in parallel job scheduling evaluation," in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2005.

[11] D. G. Feitelson, "Metrics for parallel job scheduling and their convergence," in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2001.

[12] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Parallel job scheduling for power constrained HPC systems," *Parallel Computing*, 2012.

[13] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of overprovisioned hpc data centers under a strict power budget," in *SuperComputing (SC)*, 2014.

[14] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. de Supinski, "Practical Resource Management in Power-Constrained, High Performance Computing," in *High-Performance Parallel and Distributed Computing (HPDC)*, 2015.

[15] N. Gholkar, F. Mueller, and B. Rountree, "Power tuning HPC jobs on power-constrained systems," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT 2016, Haifa, Israel, September 11-15, 2016*, 2016, pp. 179–191.

[16] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-Aware Scheduling for Real-Time Systems: A Survey," *Transactions on Embedded Computing Systems (TECS)*, 2016.

[17] V. Elisseev, J. Baker, N. Morgan, L. Brochard, and T. Hewitt, "Energy aware scheduling study on bluewonder," in *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing, E2SC 2016, Salt Lake, Utah, USA, November 20, 2016*, 2016.

[18] P. Murali and S. Vadhiyar, "Metascheduling of HPC Jobs in Day-Ahead Electricity Markets," in *High Performance Computing (HiPC)*, 2015.

[19] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M. E. Papka, "Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems," in *SuperComputing (SC)*, 2013.

[20] B. Khemka *et al.*, "Utility Driven Dynamic Resource Management in an Oversubscribed Energy-Constrained Heterogeneous System," in *International Parallel and Distributed Processing Symposium Workshop (IPDPS) Workshops*, 2014.
[21] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE Computer*, 2007.
[22] J. Hikita, A. Hirano, and H. Nakashima, "Saving 200kw and $200 k/year by power-aware job/machine scheduling," in *International Parallel and Distributed Processing Symposium (IPDPS) Workshops*, 2008.
[23] A. C. Orgerie, L. Lefèvre, and J. P. 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.
[24] P.-F. Dutot, M. Mercier, M. Poquet, and O. Richard, "Batsim: a realistic language-independent resources and jobs management systems simulator," in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2016.
[25] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing (JPDC)*, 2014.
[26] D. Balouek *et al.*, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science (CLOSER)*, 2013.
[27] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *Journal of Parallel and Distributed Computing (JPDC)*, 2014.
[28] R. D. Morey, "Confidence intervals from normalized data: A correction to cousineau (2005)," *Reason*, vol. 4, no. 2, pp. 61–64, 2008.
[29] J. Eastep, "An overview of geo (global energy optimization)," https://eehpcwg.llnl.gov/documents/webinars/systems/120915_eastep-geo.pdf.