Scheduling-based power capping in high performance computing systems

Scheduling-based Power Capping in High Performance Computing Systems

Andrea Borghesi[a,b], Andrea Bartolini[b,c], Michele Lombardi[a], Michela Milano[a], Luca Benini[b,c]

[a] *DISI, University of Bologna. Viale Risorgimento 2, 40123, Bologna, Italy*
[b] *DEI, University of Bologna. Viale Risorgimento 2, 40123, Bologna, Italy*
[c] *Integrated Systems Laboratory at ETH Zurich, Switzerland*

Abstract

Supercomputer installed capacity worldwide increased for many years and further growth is expected in the future. The next goal for High Performance Computing (HPC) systems is reaching Exascale. The increase in computational power threatens to lead to unacceptable power demands, if future machines will be built using current technology. Therefore reducing supercomputer power consumption has been the subject of intense research. A common approach to curtail the excessive power demands of supercomputers is to hard-bound their consumption, *power capping*. Power capping can be enforced by reactively throttling system performance when the power bound is hit, or by scheduling workload in a proactive fashion to avoid hitting the bound. In this paper we explore the second approach: our scheduler meets power capping constraints and minimizes Quality-of-Service (QoS) disruption through smart planning of the job execution order. The approach is based on Constraint Programming in conjunction with a Machine Learning module predicting the power consumptions of HPC applications. We evaluate our method on the Eurora supercomputer, using both synthetic workloads and historical traces. Our approach outperforms the state-of-the-art power capping techniques in terms of waiting time and QoS, while keeping schedule computation time under control.

Keywords: Constraint Programming, Optimization, HPC, Power Consumption, Scheduling, Machine Learning, Power Modeling

1. Introduction

Power consumption is a critical limiter for next generation High Performance Computing systems: supercomputers are expected to reach Exascale in 2023 [1], as revealed by the increase of the worldwide supercomputer installation [2], but at the price of unsustainable power demand growth. Today's most powerful system is Sunway TaihuLight which reaches 93 PetaFlops with 15.371 MWatts of power dissipation [3]. An Exascale machine built with current technology would consume an excessive amount of power (hundreds of MWatts), while a commonly accepted upper bound for a supercomputer power consumption is around 20MW [4]. Therefore, in the last years the HPC community put great effort in finding effective ways to reduce power consumption of HPC facilities, either developing new hardware and software solutions or optimizing the management of existing systems.

Many strategies try to limit the power consumption within a certain power budget, never to be exceeded. These methods are generally referred to as *power capping* [5]. A big challenge for the adoption of power capping solutions is the need to find a good balance between curtailing power consumption and keeping a high level of Quality-Of-Service for the system users. In particular, most power capping approaches rely on decreasing the performance of the computing nodes (i.e. imposing a node power budget and/or decreasing the operational frequency as a reactive countermeasure when the power cap is reached) which in turn leads to increased duration of HPC jobs or longer waiting times. Increased durations can produce user dissatisfaction due to longer completion times and possibly increased costs. In fact the accounting policy in some current supercomputers links the price for the user to the duration (number of CPU-hours) of the job.

We reckon that a key role in this challenge can be played by the scheduling software that decides where and when a job has to execute, a software module commonly referred to as *job dispatcher*. We claim that with a "clever" job dispatcher it is possible to operate a power capped system at a higher Quality-of-Service than using reactive performance throttling techniques that are in common use today. Constraint Programming (CP) is a paradigm to solve NP-hard problems by exploring a set of feasible solutions optimizing one or multiple objective functions. However, this technique is not widely used in HPC facilities, because the run-time of solvers are not compatible with the online nature of job schedulers for supercomputers. On the other hand, supercomputer jobs do have a significant duration and their arrival rate is significantly lower than that of e.g. data centers and enterprise servers workload. This creates opportunity for an optimization-based scheduling. The feasibility of this approach has been already demonstrated by several works [6, 7].

In this paper we propose a job dispatcher able to limit a supercomputer power consumption acting only on the workload scheduling. Our approach satisfies the power constraint work-

Email addresses: andrea.borghesi3@unibo.it (Andrea Borghesi), a.bartolini@unibo.it, barandre@iis.ee.ethz.ch (Andrea Bartolini), michele.lombardi2@unibo.it (Michele Lombardi), michela.milano@unibo.it (Michela Milano), luca.benini@unibo.it,luca.benini@iis.ee.ethz.ch (Luca Benini)

ing on the job execution order and minimizes QoS degradation thanks to proactive planning. We used the Constraint Programming methodology to develop a *proactive* job scheduler: we adopt a rolling horizon approach, where our scheduler is awakened at certain events. At each of such activations, we build a full schedule and resource assignment for all the waiting jobs, but then we dispatch only those jobs that are scheduled for immediate execution. By taking into account forthcoming jobs, we avoid making dispatching decisions with undesirable consequences; by starting only the ones scheduled for immediate execution, the system can manage uncertain execution times. An essential requirement for such a dispatcher is the knowledge of job power consumptions at schedule-time. Therefore we also developed a prediction model to estimate the power consumption of each job before its execution; this predictor relies on Machine Learning methodologies.

The contributions of this paper are the following:

1. A job dispatcher which combines a module to predict power consumptions and a job scheduler able to run HPC applications while bounding the overall power consumption.
2. An analysis of the impact of prediction errors.
3. A hybrid dispatcher combining our proactive scheduling with reactive power management from the literature.
4. A comparison between our method and several approaches from the current state-of-the-art; the experiments reveal that our method performs extremely well, with an average improvement of 8.5% w.r.t. the best state-of-the-art techniques.

The paper is organized as follows: we first discuss related works in Section 2, then we introduce the exact scheduling and allocation problem considered in this paper in Section 3. Section 4 presents the job dispatcher with power cap and the related power consumptions predictor. Finally in Section 5 we discuss the results of the comparison between our methods and other approaches for power capped job dispatchers described in the literature.

2. Related Works

Since the HPC community widely recognizes the need to reduce power consumption in supercomputers, several research avenues have been explored for this purpose, in particular power capping solutions. Many techniques have been proposed to bound the power consumption of HPC machines, ranging from Dynamic Voltage and Frequency Scaling (DVFS) [8, 9, 10, 11, 12], energy proportional systems [13], overprovisioning [14], turning off idle resources [15], exploiting components variability [16] , optimizing the placements of jobs and tasks in order to reduce communication costs [17]. In the rest of this Section 2.1 we are going to discuss power capping methods found in the literature.

An orthogonal but strictly related research direction explores the possibility to predict the power consumption (or other metrics) of HPC applications. Knowing jobs powers before their execution may lead to several benefits, such as taking well-informed scheduling decision, robust algorithm, power savings. We are going to review power prediction methodologies in Section 2.2.

2.1. Power Capping methods

We divide the power capping approaches in four different (and quite broadly defined) areas: 1) techniques which exploit some system related characteristics (i.e. node variability) or make assumptions on the nature of the workload; 2) techniques employing some form of frequency scaling; 3) techniques employing Intel's *Running Average Power Limit* (RAPL) [18]; 4) techniques which use heuristic or proactive algorithms and works on the job execution order.

2.1.1. System/workload-dependent techniques

In [19], Sarood et al. describe an ILP model to enforce power capping in a HPC cluster through over-provisioning. Their approach combines over-provisioning with a power-aware scheduler. However, this work focuses on data centers and is based on assumptions that do not hold in typical HPC workloads. For example, the proposed method requires to change the number of nodes used by a job during its execution. In the majority of today's HPC environments this is not possible yet[1], since resources are locked to a job for its entire runtime. Mammela et al. [20] present an energy-aware scheduler that can be applied to HPC systems without any changes in hardware components. The idea is to turn off idle nodes whenever possible, that is every time the scheduler detects that no activity can be scheduled for a sufficiently long time on a certain node. The main drawback of this approach is the fact that it strongly depends on the possibility to turn off idle nodes, which cannot be taken for granted in every HPC system and usually has non negligible associated costs (i.e. reliability loss and wake-up time).

Several approaches use the possibility offered by "moldable jobs", i.e. jobs which can run with different configurations (number of nodes, cores or threads) [21]: given the current power consumption and power budget, the best configuration is chosen for each job before its start - the configuration will not change during the execution. For example Patki et al. [22] propose a job dispatcher that operates on top of a EASY-BF [23] scheduler and decides the best configuration of the application to be run. The choice is based on the predicted power consumption and duration for each combination of application-configuration. While these techniques proved to be effective, their dependence on the moldable job model limits their adoption. Furthermore, deciding the best configuration requires to perform multiple off-line runs of each application in all possible configurations, to understand the configurations power-performance trade-offs. Other authors tried to exploit the power and performance variability among nodes and components within the same system. For example Inadomi et al. [24] analyze the impact of differences introduced by the manufacturing process in power-capped supercomputers; they prove

[1] Current research activities in malleable jobs are working in this direction

that a job scheduler which takes into account such differences can obtain a speed up w.r.t. an unaware scheduler. Shoukourian et al. [16] devise a power-aware configuration adviser that tries to dispatch the job on a supercomputer minimizing the total energy consumption. The main limitation of such methods is their reliance on system/component-specific characteristics, which prevent their use on different machines.

2.1.2. Frequency Scaling techniques

Nowadays most of power constrained supercomputers employ some form DVFS, i.e. they exchange processor performance for lower power consumption. With DVFS, a processor can run at one of the supported frequency/voltage pairs lower than the nominal one. The main issue with DVFS-based approaches is the trade-off between power savings and decrease in performance: reducing the operating clock clearly increases the duration of the applications which run on the power-reduced resources. To overcome this issue, several methods try to apply DVFS only in periods of low system activities or in particular phases of a job execution. For example, in [25], Freeh et al. study the energy-time trade-off of high performance cluster nodes with several power states available. They conclude that applying DVFS to applications with memory or communication bottlenecks does not imply large time penalties. A clear weakness of this strategy is that it strongly relies on the nature of the running applications, which must be known and modeled in advance, before their actual execution. In [26] Hsu et al. propose to solve this problem through a power-aware *adaptive* algorithm which does not employ any application-specific information a priori, but implicitly gathers such information at run-time. Unfortunately, this requires specific monitoring instruments to collect the required information and these tools are not currently available in the majority of supercomputers.

A method to extend the EASY-backfilling algorithm with power budgeting capability through frequency scaling is discussed by Etinski et al. [27]. The results show good results in terms of energy savings and also a better utilization of the system and reduced waiting time for the users, thanks to the possibility to execute more jobs concurrently if their frequency (thus power) is reduced. In [28], Etinski et al. propose an approach which tries to minimize the performance loss by reducing the frequency only when the system is in a low level of utilization. They combine the EASY backfilling algorithm as scheduling policy with a frequency assignment module. When a new job is dispatched the system assigns an operational frequency to its execution node (such frequency will be kept for all the job lifetime); this frequency may be lower than the maximal one depending on the current level of workload. Their results show that it is possible to obtain good energy savings without decreasing the overall performance of the system. The same authors present also a different approach in [29]: in the latter work they propose a novel scheduling policy based on integer linear programming (ILP). Instead of relying on the backfilling algorithm the new scheduler decides which jobs need to be executed and the optimal frequency solving a optimization problem. This method offers better performance in terms of average job wait time over various power budget, but it has the disadvantage of having been tested on a relatively simple case-study (i.e. homogeneous resources) and the corresponding ILP model cannot directly extended to different kinds of problems.

Kumar et Al. [30] employs DVFS to create an energy-aware scheduler for heterogeneous data centers. The main idea is to use frequency scaling during slack periods, that is when the system workload is relatively low. The proposed scheduling approach is promising but it is not directly comparable to ours due to the different characteristics of data centers w.r.t. to HPC machines.

2.1.3. RAPL-based techniques

An alternative to direct frequency scaling is RAPL, which provides a software configurable and hardware enforced power cap. Instead of setting a specific frequency, this mechanism takes as input the power budget for a socket and subsequently forces the power consumption to be within the limit. Methods using this mechanism usually employ simple scheduling and allocation policies (such as EASY-BF and First-Fit); at run time the power consumption are measured and kept under control. For example, Bodas et al. propose a power-aware scheduler [31] that decides the power available to each node of the system depending on the current power consumption and power budget. The effectiveness of this method is limited by the simplicity and rigidity of the rules that assign the power to the nodes. Ellsworth et al. [32] present a more complex scheme to decide the power allocated to each node - which they call *Dynamic Power Sharing*. Initially the overall available power budget is uniformly divided among all nodes; periodically the algorithm adjusts the allocated power depending on actual consumptions, i.e. if a node consumes less power than the allocated one the excess capacity can be transferred to a different node which needs it. RAPL is used to enforce the node power limit at run time. The main drawback of these techniques is the same that troubles DVFS mechanisms, namely the indiscriminate power reduction implies an increase in job duration (performance loss).

2.1.4. Proactive/Heuristic techniques

Another strategy to impose a power constraint is to act on the job execution order alone, without requiring any hardware modification nor any change in the operational frequencies of the computing nodes. Most of the research works belonging to this category are more focused on minimizing the energy consumption and/or the related energy costs [33, 34, 35, 36] rather than enforcing a power cap. Furthermore, they are more related to data centers and federated grids field, therefore the proposed techniques are not directly comparable with our approach to curtail the power consumption of HPC machines under a threshold.

A first method to enforce power capping through job ordering is to modify the common EASY-backfilling algorithm (EASY-BF) [23], a policy used in many real systems due to its simplicity and scalability. The algorithm tries to fit as many jobs as possible on the system (selecting them from a FIFO queue); a job fits in the system if enough resources are available. In order to add power awareness it is sufficient to consider the power as an additional resource - a job fits in the system only if its

power consumption will not cause the overall power to exceed the given budget.

A different approach requires to devise a *proactive* dispatcher, i.e. able to plan in advance the execution of all the activities to be run (taking decision for all the jobs in the waiting queue). The dispatcher can then solve an optimization problem to find the best scheduling and allocation while respecting the power constraint. In this approach the dispatcher needs to have information about the power consumption of the tasks to schedule in order to take the correct decisions. Since these power information are needed at schedule time, before the execution of the job, power consumption estimates are required. In the rest of the paper we are going to discuss a job dispatcher of this kind.

Li provides [37] an example of how the dispatching process can influence a supercomputer power and energy efficiency. The author presents a heuristic algorithm to compute an allocation of tasks on a system composed by a set of heterogeneous resources. The algorithm does not focus on power consumption but rather aims at constraining the energy consumption or, alternatively, minimizing the workload execution time. The core of the algorithm is the identification of the optimal resource partitions guaranteeing minimal energy consumption. The main limitation of the proposed technique is that it does not consider multiple goals, namely it can optimize either the energy consumption or the execution time.

2.2. Predicting Power Consumptions

As in our approach we want to obtain a dispatcher able to enforce power capping without employing real-time correction system such as RAPL. To do that we must *a priori* generate job schedules that will never exceed the power budget - at least within a confidence range. We therefore need a mechanism to estimate the future power consumption of a job using the information available at schedule-time. The importance of such predictions was underlined by several works [38, 39, 40]. Furthermore, a greater prediction accuracy is related to a better performance of a power capped dispatcher (in terms of higher machine utilization and greater energy savings) [41]. Intuitively if we could know the exact power consumption of each application we could generate optimal schedules; conversely, imperfect estimates may lead to possibly infeasible solutions.

A common way to estimate an application energy or power consumption exploits hardware performance counters which monitors the system's components usage during the workload execution [42, 43, 44]. Despite the good accuracy obtained with these models the need to know the performance counters - to be measured at runtime - clashes with the idea of having power consumption predictions available during the dispatching phase. A model to predict energy and power consumptions is presented by Shoukourian et al [45]. The suggested approach does not require any application code instrumentation and allows power and energy consumption prediction. The main limit of the described method is that it considers only jobs that use entire computational nodes (this is due to the characteristics of the considered supercomputer). This on one hand simplifies the power consumption prediction but on the other hand cannot be directly generalized to different systems where multiple applications can possibly concurrently run on the same node.

Auweter et al [46] propose an energy aware scheduler to reduce energy consumption of supercomputers. For this purpose they introduce a prediction model to forecast power and performance of HPC applications. This model heavily relies on precise information about the application executables and requires the user to provide a tag identifying similar jobs. While we think this is an interesting direction, currently users provided information cannot be taken for granted.

3. Preliminaries

We are now going to describe the context of the job dispatcher presented in the paper and to give a quick glance at an essential enabling technology. Section 3.1 details the job dispatching problem in HPC systems; Section 3.2 presents the supercomputer that served as our case-study. Section 3.3 introduces the Constraint Programming methodology.

3.1. The HPC Dispatching Problem

Dispatching jobs in a HPC system can be described as follows. We have a set of jobs $J = \{j_1, ..j_{NJ}\}$. Every job $j_i \in J$ enters the system at a certain arrival time eqt_i, by being submitted to a specific queue (depending on the user choices and on the job characteristics), $q_h \in Q$ where $Q = \{q_1, .., q_m\}$. Each queue is characterized by its expected waiting time ewt_h, which provides a rough indication of the queue priority. Each job specifies a maximal expected duration (its wall-time) d_i. Each job is composed by a set of sub-units; the number of job units of job i is u_i. Each job unit starts and ends with the job, and requires a certain amount of resources.

HPC machines are composed by sets of nodes $N = \{n_1, .., n_{NN}\}$ and sets of resources $R = \{r_1, .., r_{NR}\}$, i.e. cores, GPUs and MICs. Each node $n_j \in N$ has a capacity cap_{jr} for every resource $r \in R$. If a resource is absent from a node the corresponding capacity is zero. Each job unit k of job i requires an amount of resource $req_{ikr}, \forall r \in R$. Formally, the power cap can be seen as an additional constraint on an artificial resource, the power consumption, that must never be violated; we assume that each job unit has a fixed power consumption.

The dispatching problem at time τ requires to assign a start time $st_i \geq \tau$ to each waiting job i and a node to each of its units. All the resource and power capacity limits should be respected, taking into account the presence of jobs already in execution. Once the problem is solved, only the jobs having $st_i = \tau$ are actually dispatched. The single activities have no deadline or release time (i.e. they do not have to end within or start after a certain date), nor the global makespan is constrained by any upper bound.

3.2. EURORA System

The Eurora supercomputer, developed by Eurotech and Cineca [47] has ranked first in the Green500 list in July 2013, achieving 3.2 GFlops/W on the Linpack Benchmark with a

peak power consumption of 30.7 KW. Eurora has been supported by PRACE 2IP project [48] and serves as testbed for next generation Tier-0 system. Its outstanding energy efficiency is achieved by adopting a direct liquid cooling solution and a heterogeneous architecture with best-in-class general purpose hardware components (Intel Xeon E5, Intel Xeon Phi and NVIDIA Kepler K20). The system was hosted at the Cineca inter-universities consortium [49] facilities in Bologna, Italy. As described in [50] Eurora has a heterogeneous architecture based on nodes (blades). The system has 64 nodes, each with 2 octa-core CPUs and 2 expansion cards configured to host an accelerator module: currently, 32 nodes host 2 powerful NVidia GPUs, while the remaining ones are equipped with 2 Intel Xeon Phi accelerators. Every node has 16GB of installed RAM memory.

Jobs are submitted by the users into one of multiple queues, each one characterized by different access requirements and by a different estimated waiting time. Users submit their jobs by specifying 1) the number of required nodes; 2) the number of required cores per node; 3) the number of required GPUs and Xeon Phi per node (never both of them at the same time); 4) the amount of required memory per node; 5) the maximum execution time. In other HPC systems the users might not be required to specify a queue at submission time; in this case it would be trivial to implement a preprocessing phase that sends different jobs to different queues (i.e. according to the requested resources) before the job scheduling phase that is our concern in the paper.

3.3. Constraint Programming

Constraint Programming (CP) is a declarative programming paradigm suitable for solving constraint satisfaction and optimization problems[51]. A constraint program is defined on a set of decision variables, each ranging on a discrete domain of values that the variable can assume, and a set of constraints limiting the combination of variable-value assignments. For example, variable x with domain [1..10] means that variable x can be assigned to one integer value between 1 and 10.

After the creation of the model, the CP solver alternates two main phases. First, constraint propagation: constraints are propagated by removing provably inconsistent values from variable domains. The constraint $x < y$ where both x and y domains are [1..10] removes value 10 for x and 1 for y. Second, the search strategy explores alternative assignments of variable-values until either a solution is found or a failure is detected. In case of optimization problems, finding a solution does not guarantee its optimality. The solver then looks for better solutions if they exist, otherwise it proves optimality.

Historically, CP techniques have shown great success when dealing with scheduling and resource allocation problems, thanks to the expressive and flexible language used to model the problem and powerful algorithms to quickly find good quality solutions[52]. Activities are modeled through decision variables; each activity a is characterized by its start time $st(a)$ and its duration $d(a)$.

The CP community has developed a class of particularly powerful constraint called *global* constraint, whose strength relies on efficient propagation algorithms tailored on specific problems. For scheduling problems many specialized global constraints have been developed, the most important being the cumulative constraints for managing resource usage. $cumulative([a],[r],C)$ holds if and only if all activities in $[a]$ whose resource requirements are specified in $[r]$ never exceed the resource capacity C. Several propagation algorithms are embedded in the cumulative constraint for removing inconsistent assignments of activity start time variables.

4. The Dispatcher Model

In this section we describe our job dispatcher with power capping. An earlier version of our approach was already part of previous works [53, 54]. We developed two different dispatchers: 1) a heuristic algorithm, 2) a hybrid method which decomposes the problem and uses both Constraint Programming and a heuristic technique. The first method is faster while the second method manages to find the best solutions. A key point of our approach is that it requires the power consumption estimate for each job to be scheduled; this knowledge must be know *before* the actual job execution. The method we use to generate these estimates is discussed in the next section.

4.1. Power Predictor

In our dispatcher we employ a power consumption prediction model that we previously developed [55]. In order to be of any use, a fundamental requirement for this prediction model is the capability to estimate power consumption depending only on the information available at schedule time (and not at execution time, such as sensors measurements). Therefore, our model can estimate the power consumptions of HPC applications using only the information provided by the users when they submit their job in the system. Namely, the information needed to make a prediction are the following: user, queue, requested duration, number of requested nodes, number of requested cores, number of requested GPUs, number of requested Xeon Phi[2], amount of requested memory.

The model predicts exclusively the CPU power consumption of HPC applications. We assume the power consumption of additional resources (GPUs, MICs, etc.) is directly proportional to the amount of resources used and their TDP[3]. For example if a job requires 2 GPUs, its estimated power consumption is the sum of the CPU power predicted with the prediction model and the maximum power consumption of the required GPU multiplied by two. An important aspect of the prediction model is that it returns a single value, that is the mean power consumption of a job over its lifetime. As demonstrated in [55] this is not a limitation: although each job power consumption is not constant when we consider a realistic workload composed by

[2]The number of requested hardware accelerators is important because GPUs adn Xeon Phi are mounted on computing nodes with different power consumptions, i.e. a job requiring a GPU will necessarily run on a CPU consuming more power than those with a Xeon Phi

[3]Thermal Design Power

many jobs the sum of their mean powers closely tracks the system overall power consumption.

The predictor relies on Machine Learning techniques and it was trained and later tested using historical job traces and sensors data gathered in the Eurora system. The essential steps are the following. 1) We create a training set: using historical data (18 months of Eurora life time) we are able to associate a power consumption to each job that executed on the supercomputer. Part of the data is kept to be used for testing – as it is mandatory practice in ML, training and test sets do not overlap. 2) We train several prediction models (one for each user plus one for users who are submitting jobs for the first time) in order to learn the association between a job request and its power consumption. We used prediction models based on Decision Trees Regression [56] and Random Forests Regression [57]. 3) After the training phase we try the models on the test set: we predict the power for every job in the test set and then we compare the prediction and the real value.

The mean absolute error over the whole training set is 8.87%. In Figure 1 we can see some results; the figures correspond to a two-days period[4]. Figure 1a compares the predicted trend to the real power trend and Figure 1b displays the histograms of the prediction errors. For this particular time span the results are very good, with a mean absolute error smaller than 6%.

An important thing to point out is that our prediction model might be inaccurate. This issue has an obvious impact on our approach: since our dispatcher relies on the power predictions in order to generate schedule that will never exceed the power budget, mispredictions may lead to situations where the power cap enforced differs from the desired one. This can happen especially in case of *under-predictions*, when the real power actually exceeds the predicted value and consequently the cap planned in the dispatcher.

Our dispatcher (as well as all the others that rely on estimating power consumption to decide a schedule) can guarantee not to violate the power budget only within a certain confidence interval, given by the accuracy of the prediction. There are several ways to cope with this issue. On one hand, we can back up our dispatcher with a hardware-based mechanism to ensure to never exceed the reserved power budget. Another solution could be to require the dispatcher to respect a power constraint tighter than the real one, thus guaranteeing never to surpass the desired power budget. We evaluate the impact of safety margins in Section 5.2.4.

4.2. Heuristic Approach

The first approach belongs to a class of scheduling techniques known in the literature as *Priority Rules Based* scheduling (PRB)[58]. The main idea is to order the set of tasks to be scheduled, constructing the ordered list by assigning priority for each task. Tasks are selected according to their priorities and each selected task is assigned to a node; even the resources are ordered and the ones with higher priority are preferred - if available. This is a heuristic technique and it is not able to guarantee an optimal solution but has the great advantage of being extremely fast. More details on the algorithm can be found in [54].

The jobs are ordered w.r.t to their expected waiting times, with the "job demand" (job requirements multiplied by the job estimated duration) used to break ties. Therefore, jobs which are expected to wait less have higher priority, subsequently jobs with smaller requirements and shorter durations are preferred over heavier and longer ones. The mapper selects one job at time and maps it on a available node with sufficient resources. The nodes are ordered using two criteria: 1) at first, more energy efficient nodes are preferred (i.e. cores that operate at higher frequencies also consume more power) 2) in case of ties, we favour nodes based on their current load (nodes with fewer free resources are preferred[5]).

The PRB algorithm proceeds by iteratively trying to dispatch all the activities that need to be run and terminates only when there are no more jobs to dispatch. We suppose that at time $t = 0$ all the resources are fully available, therefore the PRB algorithm starts by simply trying to fit as many activities as possible on the machine, respecting all resource constraints and considering both jobs and nodes in the order defined by the priority rules. Jobs that cannot start at time 0 are scheduled at the first available time slot. At each time-event the algorithm tries to allocate and start as many waiting jobs as possible and it will keep postponing those whose requirements cannot be met yet.

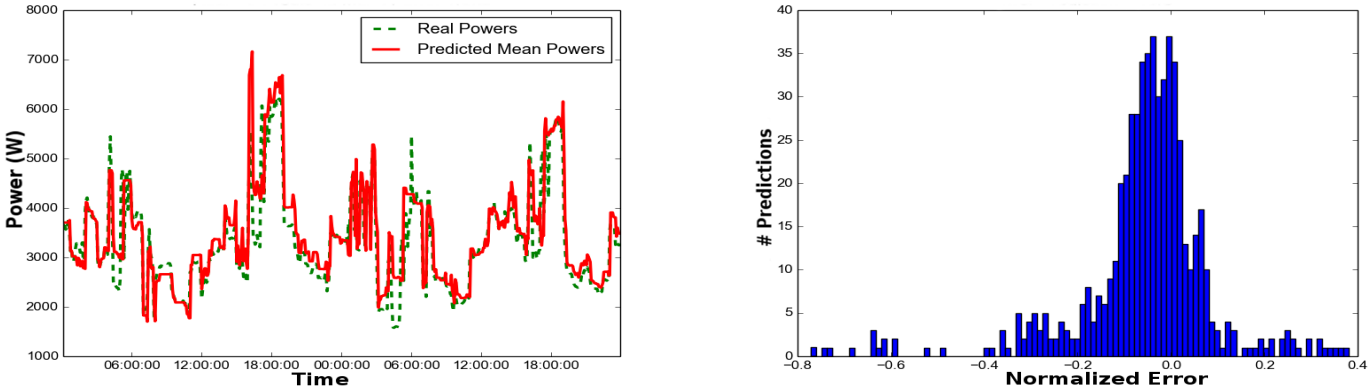
The algorithm considers and enforces constraints on all the resources of the system, including power. A job can be scheduled in the machine if there are enough available physical resources (such as cores or GPUs) and if adding its predicted power to the current system consumption would not cause a violation of the power budget. We note that since in this problem we have no deadline on the single activities , the PRB algorithm will always find a feasible solution, for example delaying the least important jobs until enough resources become available due to the completion of previously started tasks.

4.3. Hybrid Approach

The task of obtaining a dispatching plan on Eurora can be naturally framed as a resource allocation and scheduling problem. We decompose the dispatching problem in two stages: 1) obtain a schedule using a relaxed CP model of the problem 2) find a feasible mapping using a heuristic technique. Since we used a relaxed model in the first stage, the schedule obtained may contain some inconsistencies; these are fixed during the mapping phase, thus we eventually obtain a feasible solution, i.e. a feasible allocation and schedule for all the jobs. To make this interaction effective, we devised a feedback mechanism between the second and the first stage, i.e. from the infeasibilities found during the allocation phase we learn new constraints that will guide the search of new scheduling solutions at following iterations. This two stages are repeated n times, where n has

[4] We used a one month long interval as a test set but the resulting plot would not be readable

[5] This criterion should decrease the fragmentation of the system, trying to fit as many job as possible on the same node



(a) Predicted VS Real Power

(b) Error Histogram

Figure 1: Comparison between the real total power and the predicted total power. Mean Absolute Error: 0.056. Figures taken from [55]

been empirically chosen after an exploratory analysis, keeping in mind the trade-off between the quality of the solution and the computational time required to find one.

We implemented the power capping requirements as an additional constraint: on top of the finite resources available in the system such as CPUs or memory, we treat the power as an artificial resource with its own fixed capacity (i.e. the user-specified power cap), which we cannot "over-consume" at any moment.

The scheduling problem consists in deciding the start times of a set of jobs $i \in J$ satisfying the finite resource constraints and the power capping constraint. Since all the job-units belonging to the same jobs must start at the same time, during the scheduling phase we can overlook the different units since we need only the start time for each job. Whereas in the actual problem the resources are split among several nodes, the relaxed version we use in our two-stages approach considers all the resources of the same type (cores, memory, GPUs, MICs) as a pool of resources with a capacity Cap_r^T which is the sum of all the node resource capacities, $Cap_r^T = \sum_{j \in N} cap_{k,r} \quad \forall r \in R$. As mentioned before the power is considered as an another resource type of the system, so we have a set of indexes R' corresponding to the resource types (cores, memory, GPUs, MICs plus the power); the overall capacity Cap_{power}^T is equal to the user-defined power cap.

The CP model represents every job as an activity τ. Each activity is defined by its start time $s(\tau)$ and its end time $e(\tau)$; the duration is $d(\tau)$. The activity may or may not be present; if not present it does not affect the model). The activities can be subject to several different constraints, among them the `cumulative` constraint[59] to model finite capacity resources.

$$\forall r \in R' \quad cumulative(A, req_r, Cap_r^T) \tag{1}$$

where A is the vector with all the interval vars, where req_r are the job requirements for resource r (for all jobs in A); the job power consumptions are those predicted as mentioned in Section 2.2. The cumulative constraints in 1 enforce that at any given time, the sum of all job requirements will not exceed the available capacity (for every resource type).

With this model it would be easy to define several different goals, depending on the metric we optimize. Currently we use as objective function the *weighted queue time* [60, 61], i.e. we want to minimize the sum of the waiting times of all the jobs, weighted on estimated waiting time for each job (greater weights to job which should not wait long):

$$\min \sum_{i \in I} \frac{\max ewt_i}{ewt_i}(s(\tau_i) - q_i) \tag{2}$$

where $s(\tau_i) - q_i$ represents the waiting time (the time when a job begins its execution minus the time when it was submitted) and $\frac{\max ewt_i}{ewt_i}$ is a coefficient that serves to giver higher priority to job that should wait less. The Expected Waiting Time of a job ewt_i depends on its queue and lower values indicate that a job should have a higher priority; $\max ewt_i$ represent the expected waiting time of the queue with lower priority. Hence, the ratio $\frac{\max ewt_i}{ewt_i}$ has bigger values for jobs with higher priority (smaller denominator) and thus increases the weight of these jobs.

To solve the scheduling model we implemented a custom search strategy that begins from the solution obtained by the heuristic algorithm (described in Section 4.2) and then tries to improve it (shifting the starting time of the job variables towards the origin, thus "squeezing" the resulting schedule). The time limit allowed to explore the solutions space may vary: we first try to find a solution within 1 second, if the time is not sufficient (the complexity grows exponentially with the number of jobs) we double the time limit. The maximal time limit is one minute. In practice, since the search starts from the heuristic solution a first solution is always found within the first 10 seconds.

5. Experimental Evaluation

After having described our approach, in this Section we compare it to state-of-the-art methods. In order to make such a comparison we created a simulation framework to implement all dispatchers. The simulator takes as an input an instance composed by a set of job requests (user name, job id, resource requested, etc.) and then the chosen dispatcher takes the scheduling and allocation decisions. Applications fall in one of three

possible categories: CPU-bound, memory-bound, and mixed (average workload slightly skewed toward CPU usage).

The job instances used are both historical job traces which ran on the Eurora system and synthetic benchmarks, created following the traces. The features of each job (its duration, requested resources, etc.) are drawn from random distributions defined by parameters learned from past workloads. Each job in an instance has an arrival time, i.e. the moment when it enters the system; the arrival times are distributed within a time window. The arrival times distribution can follow three different models: 1) uniform distribution; 2) left-skewed distribution (a larger fraction of the jobs starts close to the window origin); 3) grouped distribution (also referred as *burst*), where jobs arrive in groups. For example a job instance might be composed by 400 jobs that enter in the system in 15 minutes.

5.1. State-of-The-Art Comparison

We selected a sub-set of the methods from the State-of-the-Art for the comparison with our approach. First we excluded all methods based on moldable or malleable jobs since we consider only rigid ones[6]. We also disregarded methods relying on system-specific features, i.e. node variability. Finally, our dispatcher deals with a non-trivial problem (multi-resource system, jobs composed by sub-units, etc) and therefore we did not considered approaches not applicable to such problem. For example we implemented [27] instead of a more recent technique from the same authors [29] because the latter would have required to develop an entirely different ILP model than the one presented in their paper.

The following are the chosen methods. Our two approaches: I) the heuristic algorithm *LS* (Section 4.2) and II) the hybrid approach *DEC* (Section 4.3). III) The power-aware EASY-backfilling extension described in Section 2 - referred to as *BF*. IV) A technique employing frequency scaling based on [27] - referred to as *DVFS*. Two methods relying on RAPL: V) *Simple* power-aware scheduler presented in [31] and VI) a dynamic power-sharing methods discussed in [32] - referred to as *DynShare*.

5.1.1. Impact of the power reduction/frequency scaling

A key aspect of some implemented models is the possibility to reduce the power consumption in a node via RAPL or frequency scaling (*DVFS*, *Simple*, *DynShare*). As the comparison between the different policies is done on the hardware testbed and we focus on approaches which uses RAPL and DVFS for power capping instead of energy optimization, we can remove from the power estimation the idle power of each resources which is a constant offset in the total power consumption. As result of this we consider in the following power modeling only the dynamic power component, the one dependent on the clock frequency. Since we are not considering application-level optimization, such as reducing the frequency only for memory-intensive task or tasks reordering, it is safe to assume that, in general, the duration of a "power reduced" job will increase. The amount of the change is a non-trivial issue: it depends on the hardware implementation (RAPL is a proprietary solution), the nature of the application, technological parameters, etc. In our work we tested different power-duration relation scenarios.

Let assume that we decrease the power on a certain node by a certain amount, for example we go from P to $P' = M^P P, 0 < M^P \leq 1$. The ideal case is that the duration is not affected: $D' = D$ - this is a limit case and can be used to compare a method against an idealized situation that favours to the extreme techniques that reduce power by changing operating frequency. The opposite case defines the duration increase as directly proportional to the power decrease: $D' = M^D D, M^D = 1/M^P, M^D \geq 1$. Between the best-case and this worst-case scenario (from the point of view of RAPL based solutions) we can have intermediate cases, where the duration increase is modulated by a factor F: $D' = FM^D D, M^D = 1/M^P, M^D \geq 1, 0 < F \leq 1$. This general formula can also include the first two extreme cases. Intermediate cases indirectly models non-linearities in both the power dependency with the clock frequency as well as the duration dependency on the clock frequency. In our experiments we tried the following factors: $0.25, 0.5, 0.75$. We must note that this model starts from a power reduction, neglecting the internal mechanisms with which this is obtained (DVFS, or RAPL, or throttling, ecc) to model the application slowdown. We chose a simple and yet effective linear model even if non-linearities may be present between the processor performance and power reduction[62], and between the processor performance and TtS relation[63]. By choosing F this model covers both the corner cases and the intermediate ones.

In the scenarios discussed so far we applied indiscriminately the same power-decrease/duration-increase model to all jobs. We also implemented a *mixed* model where each job has its own factor (given that we apply the more general formula). The factor of each job is drawn from one among three random distributions, one for each application type - memory-bound jobs will obtain lower factors (smaller duration increase) than the CPU intensive ones.

5.1.2. Evaluation Metric

To compare the different techniques we chose the so called *Bounded Slowdown* (BSLD), a metric commonly used in the literature [64, 28, 27, 65, 29, 66] and defined by the ratio between the time spent waiting in the system and the job runtime.

$$BSLD = max(\frac{wait\_time + run\_time'}{max(\theta, run\_time)}, 1) \quad (3)$$

where *wait_time* is the time spent waiting for execution, *run_time* is the "original" duration (i.e. the duration specified when the job is submitted), *run_time'* is the final duration (possibly changed due to power reduction or frequency scaling), θ is a threshold used to avoid the bias of very short jobs on the average value. In all our experiments we set a threshold of five minutes. This metric takes into account both the slowdown introduced by the dispatchers that enforce power capping by postponing jobs and the slowdown caused by the performance penalty due to frequency scaling. The BSLD assumes values

[6]Most current HPC systems employ rigid jobs

≥ 1 (with 1 being the optimal value); lower values indicate better performance.

In the BSLD computation we do not include the times needed to compute a schedule: the "simulated" time is not affected by scheduling delays due to the event-based nature of our simulation framework. Furthermore, even in the case of our slowest dispatching approach (*DEC*), we set a time limit of 10 seconds for the scheduling & allocation phase, which is an almost insignificant amount of time compared to duration of HPC applications. We are also going to disregard the time needed to run the simulation, although given the same set of jobs the various SoA techniques require different time: these differences are due to the different implementations and a comparison would be unfair.

5.2. Results

For every job instance we performed several experiments: first we run each dispatcher without power cap, to establish the uncapped maximal power consumption. Then we run again the dispatchers on the same instance imposing a power cap with decreasing power budget values (expressed as percentage of the uncapped power consumption). We run experiments on instances of varying size and arrivals window width, ranging from 50 jobs in five minutes to 1000 jobs in half an hour; different initial conditions were also tested, from an initially empty system to a machine where 70% of the nodes are already fully occupied. For every combination of number of jobs/time window/starting condition we ran experiments on at least 20 different job instances and gathered the corresponding evaluation metric for each run. The results displayed in the following plots are the average values of the collected statistics.

The first thing we noticed is the very poor performance of the *Simple* dispatcher; in any possible condition (low or high power budget) and any instance size this method provides schedules that are an order of magnitude worse (in terms of average BSLD) compared to the other ones. This is due to the lack of a power sharing mechanism among nodes and this leads to extremely unbalanced situations that penalize jobs that did not enter in the system as first. We decided to exclude this method from the following plots because it would have not added any significant information.

A second point we are not going to discuss further concerns the performance of our methods *LS* and *DEC*. As it was expected due to their implementation *DEC* always provides better - or equal - results than *LS*, due to the fact that they share the first solution and only *DEC* runs an additional search to improve it. When the problem grows in size *DEC* might not be able to find improving solutions (due to the tight time limit we impose) and the distance between our methods narrows.

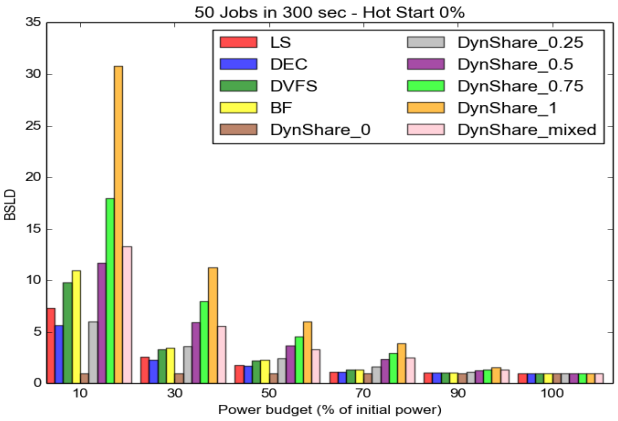
Figure 2 portrays the result obtained with instances of 50 jobs arriving in 5 minutes, uniform distribution of arrival times. In Figure 2a the machine was empty at the initial state, hot start (HS) = 0%. The power budget ranges from the maximum (100%) till a value of 10% of the maximum - we want to stress out that values lower than 40% of the unconstrained power are extremely small and quite rare in real systems. Each column corresponds to the average BSLD obtained by a dispatching method. We show our methods (*LS* & *DEC*), backfilling with power cap *BF*, frequency scaling *DVFS* and several scenarios for the RAPL methods *DynShare*; the different scenarios correspond to the different power/duration models (see Sec. 5.1.1). The number at the end of the name indicates the factor: 0 and 1 for the extreme cases (no duration increase and proportional increase); 0.25, 0.5 and 0.75 are the intermediate values; *mixed* denotes the scenario where each application has its own factor.

With such small instances the machine resources are almost never fully occupied (50 jobs for 64 nodes) and therefore the only obstacle preventing jobs to run in the system is the power constraint; this can be clearly seen noticing the small BSLD values at higher power budget - only slightly larger than one. It is clear that our methods offer very good performance: both *LS* & *DEC* perform better than all remaining methods except the *DynShare_0* at power budgets between 30% and 70%. As mentioned before *DynShare_0* represents an optimal, unrealistic lower bound (power decrease does not affect job duration and BSLD metric) and obviously performs better than all other approaches - especially when the power is the main issue. The average BSLD of the other *DynShare* techniques worsens rapidly with the increase of the power-duration factor (from 0.25 up to 1); *DynShare_mixed* has a performance close to a factor of about 0.5-0.6. As expected *DVFS* offers better results than *BF* (both better than RAPL with such small instances and empty machine). In our tests the improvement of *DVFS* w.r.t. *BF* is smaller than the one presented in [27] and this happens due to the more complex problem tackled (job and job-units versus jobs only, multi-resource machine versus single resource, nodes of different types).

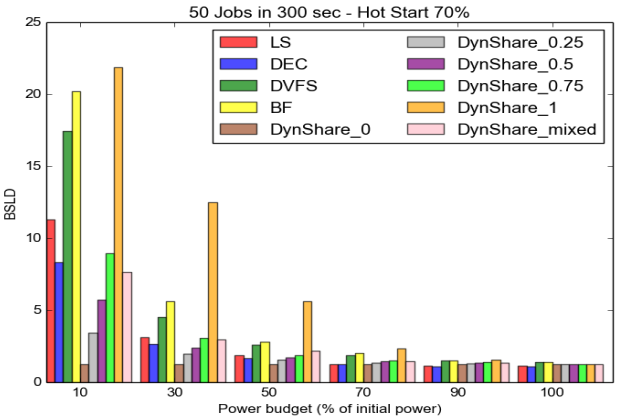
5.2.1. Initial State Impact

The situation changes when we consider a non empty machine; in Fig. 2b we can observe the results if the 70% of the nodes are occupied before the jobs arrival. Now the methods using RAPL perform much better (w.r.t. the other ones), only the case with duration increase proportional to the power decrease maintains its bad performance; *DVFS* and *BF* provide much worse results. *LS* & *DEC* manage to remain on par with all RAPL methods only until the power budget does not get too low (smaller than 30%); at a (quite unrealistic) power budget of 10% only *DEC* obtains an average BSLD close to *DynShare_mixed* and *DynShare_0.75*. If we look at the numbers we see that the differences in relative performance are due to the better results obtained by *DynShare*, while the other methods performance is worse than the empty machine case. We would expect to observe a generalized performance deterioration: in an occupied machine jobs are forced to wait until more resources become available.

To understand why this is happening we have to distinguish between two different kinds of dispatchers. RAPL-based methods are *dynamic* dispatchers, they let all jobs enter the system and then dynamically adjust their power consumptions - no power-check is performed before admitting jobs. The other methods can be seen as *static*: they decide before dispatching which jobs can be executed and their strength is based on the quality of the generated schedule. When the machine is al-



(a) Hot Start = 0%



(b) Hot Start = 70%

Figure 2: Average BSLD; 50 jobs 300s; HS=0%; uniform distribution

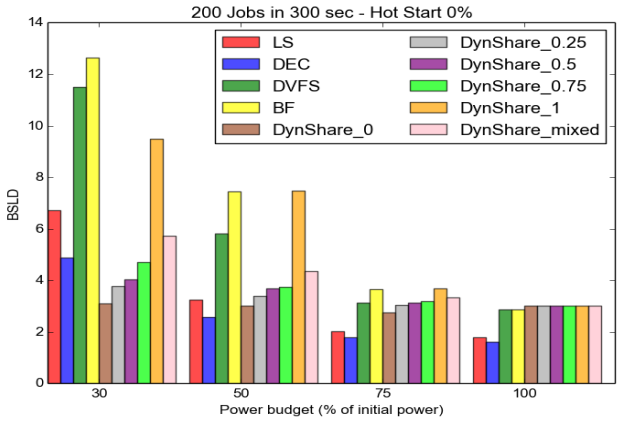


Figure 3: Average BSLD; 200 jobs in 300s; HS=0%; burst arrival

ready occupied static approaches suffer from the lack of optimization possibilities, thus reducing their effectiveness[7]. Conversely, one of the weakest point of reactive methods is the lack of a power-based admission control that may lead to too many jobs in the system and therefore widespread slowdown generated by mandatory power reductions. With a partially occupied machine fewer jobs can enter the system due to the fewer available resources (cores, GPUs, etc.), large power reductions happen less frequently and as a result the average BSLD improves.

5.2.2. *Instance Size Impact*

In Figure 3 we can observe the behaviour of the different dispatchers when the instance size increases, in particular up to 200 jobs in 300 seconds - in this case the jobs arrive in groups (burst arrival). We restrict the analysis to more realistic power budget (namely we discard the 10% case). The first thing worth to be noted is the fact that the different dispatchers obtain different results even at maximum power budget. The reason is that when the instance size increases the problem becomes harder in terms of resource availability, therefore an approach able to generate better schedules leads to better performance. In other words, even with no power constraint (power budget equal to 100%) some jobs must wait for free resource and our more sophisticated methods (*LS* & *DEC*) manage to produce lower queue times than the simple EASY-BF - hence lower BSLD. This is an extremely important point because it allows our method to outperform even *DynShare_0*, because this method (like all RAPL-based ones) focuses only in minimizing the runtime-increase penalty (see Equation 3).

With higher power budget (75%-100%) both *LS* and *DEC* outperform all remaining methods; when the power constraint gets tighter the RAPL begins to be effective and the gap with our methods is reduced - but at 50% it is still the best approach. When the power budget gets even lower (30%) the quality of the generated schedule becomes relatively less important and RAPL-based approaches take the lead - though *DEC* performance is still roughly equal to *DynShare_075* and better than *DynShare_mixed*. Both *DVFS* and *BF* generally provide worse results than the other methods.

When we increase again the size of the instance, with a number of job $\geq$ 400, we notice a clear limit of the RAPL-based methods in their simpler implementations. We can see this in Figure 4. Figure 4a displays the average BLSD for instances of 400 jobs in 300 seconds, burst arrival mode, empty initial state of the system. We see that for smaller power budgets ($\leq$ 30%) the RAPL-based methods are not able to find solutions: it is impossible to satisfy the power constraint just by applying the RAPL-mechanism. This happens for two reasons: 1) there is no power-aware job admission control - the basic EASY-BF scheduler always let jobs enter the system; 2) we assumed that there is a lower bound on the minimal power consumption of a job - we cannot arbitrarily decrease the job power below a certain threshold (1 Watt in our experiments). These two conditions (all jobs in an instance can enter the systems - provided enough resources are available - and a non-zero minimal power consumption) prevent pure reactive techniques such as the *DynShare* methods from keeping the system power consumption within the desired budget. This is not a problem for the remaining approaches due to the capability to limit *a priori* the number of jobs entering the system.

[7] *LS* & *DEC* can also be seen as *proactive* dispatchers: they take into account *all* jobs which need to be executed when creating an optimal schedule

As we identified, the problem lies in the basic implementation of the RAPL-based methods, lacking a power-aware admission control mechanism. Even though this is not discussed in the literature, we assume that real supercomputers employ some form of admission control and therefore we modified the *DynShare* methods to add such a mechanism. We therefore substituted the EASY-BF scheduler of the basic implementation with the power-aware version; in practice we combined a first scheduling stage that employs *BF* and a second stage where the power is dynamically managed with RAPL. Directly assigning the desired power budget to the *BF* algorithm makes no sense because the power cap would be already enforced and the RAPL-mechanism would never be triggered. Therefore the power budget considered in the first stage is three times the target power budget[8]; the remaining power slack is covered by RAPL action.

In Fig. 4b we can observe the results obtained after the change. Now even RAPL-based approaches provide solutions at lower power budgets. As we saw for instances composed by 200 jobs, *LS* & *DEC* clearly outperform the remaining methods when the problem is less power constrained and good scheduling decisions have a greater impact. With the tightening of the power constraint (power budget $\leq 30\%$), the possibility to decrease power consumption at run times starts to reap its benefits; nevertheless *DEC* still trails very closely *DynShare_0* and provides lower average BSLD than all other methods.

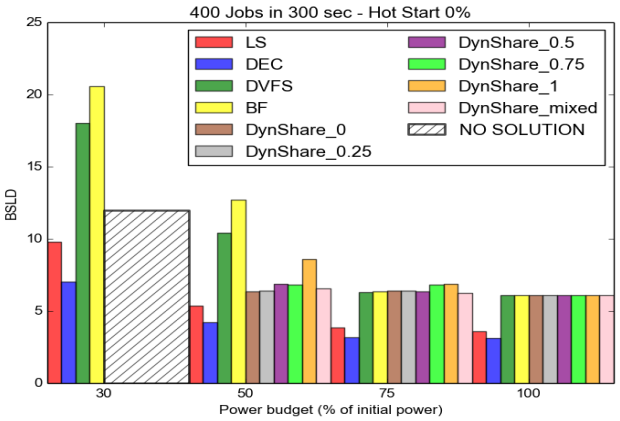
The results obtained with larger instances (up to 1000 jobs, window size ranging between 15 and 30 minutes) show an identical behaviour and thus they are not reported in the paper. The RAPL-based methods cannot find solutions without a previous power-guided admission control. When we add the admission control, *LS* & *DEC* perform equally or slightly worse (higher BSLD) than *DynShare* with lower power-performance factor (0 and 0.25). This happens only for lower budgets; again, with budgets ranging from 50% to 100% our methods clearly outperform all remaining ones (decisive advantage when the problem is less power constrained).

5.2.3. Historical Traces

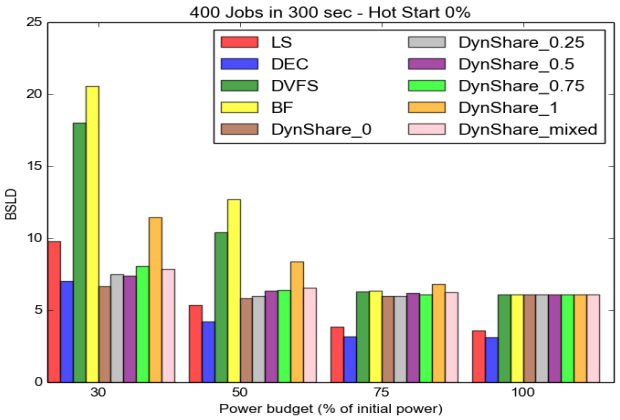
We performed experiments using real historical traces of jobs that run on Eurora. The experiment setup is the same used for the synthetic benchmark, with the only difference that in this case the arrival window and mode are defined by the trace itself.

On the historical traces we tested all previous dispatchers plus an additional type of RAPL-based approach; in particular we added a new scenario for the relationship between power decrease and duration change (see Section 5.1.1 for the other scenarios). In this case we link the duration increase to the nature of the application, defined by its Clock-Per-Instructions (CPI) value. The CPI can range between values $0 < CPI < 1$, when more than one instruction per clock cycle is executed, to $CPI >> 1$, when the execution of an instruction is bounded by the access to far memory (Last Level Cache, LLC, and DRAM) as well as in case of multi-cycle instructions – such as multiplications and divisions. In the rest of the paper we consider

[8] "Three times" is an empirically computed value



(a) No admission control



(b) Power-based admission control

Figure 4: Average BSLD; 400 jobs in 300s; HS=0%; burst arrival. Testing the importance of a power-aware job admission control for *DynShare* methods

for simplicity only the first case, as better predictions can be obtained by using CPI information in conjunction with other monitored events (i.e. LLC misses). With this assumption, we can recap the situation in this way: low values of CPI (≤ 1) indicate CPU-bound applications while higher values (≥ 5) are related to memory-bound applications; intermediate values suggest less unbalanced applications. The CPI values of the historical jobs were measured by Eurora monitoring infrastructure.

For a job of duration D executing at the maximum frequency of the core F_{MAX} the number of instructions is computed as: $\#_{INS} = D * F_{MAX}/CPI$. When applying DVFS or RAPL, the frequency decreases by a factor M^F which we assume to be equal to the target power multiplier M^P (the ratio between the new desired value P' and the original power P). With these considerations, given a value of clock-per-instructions CPI the new duration D' is:

$$\begin{aligned} D' =& \#_{INS} \frac{((CPI-1)+1/M^F)}{F_{MAX}} = \\ =& D\frac{((CPI-1)+1/M^F)}{CPI} = \\ =& D(1-\frac{1}{CPI}+\frac{1}{CPI*M^P}) \end{aligned} \quad (4)$$

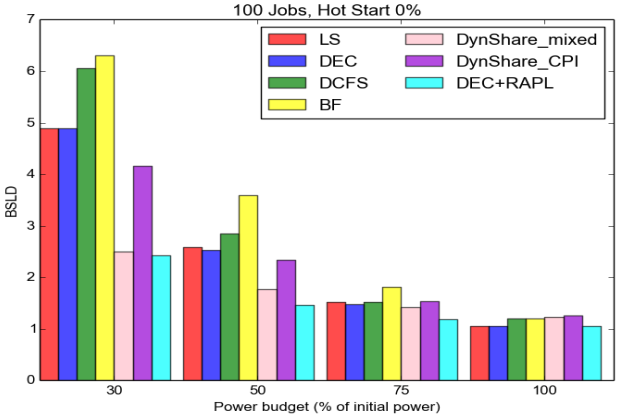
The above mentioned model assumes that the LLC and DRAM domain have an independent clock, different from the core one. This implies that when slowed down of a factor M^P the CPUs slow down only the fraction of the clock cycles executing on the core region (1 cycle if multi-cycle ALU instructions are neglected). We assume $CPI < 1$ equal to 1 as this case refers to multiple instruction executed in parallel which depends on the core speed with the same proportionality of a single instruction executed in one cycle.

The new approach for historical traces (called *DynShare_CPI*) shares the same algorithm with the other RAPL-based methods (*DynShare*) and uses the factor defined in Eq. 4 to compute the job duration change given a required power modification. With historical traces we run experiments using the following approaches: *LS*, *DEC*, *BF*, *DVFS*, *DynShare_mixed* and *DynShare_CPI*. Figure 5a portrays the average BSLD computed on 80 historical traces with a job size equal to 100; Figure 5b consider the case of 400 jobs. Both figures present an additional column *DEC+RAPL* representing an additional approach that combines *DEC* and *DynShare_mixed*; before discussing this last method more in detail we need to make a couple of observations.

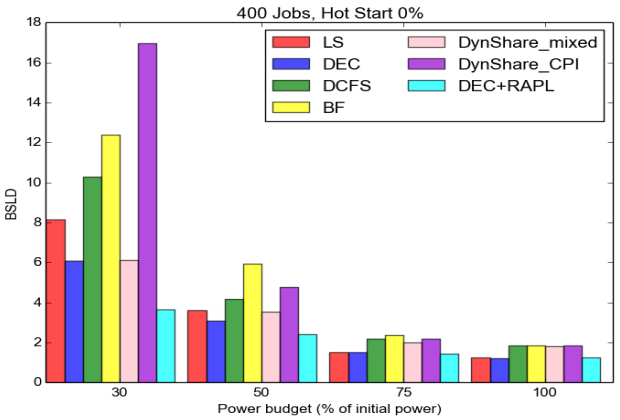
First, we notice that using the CPI-based criterion to compute the duration increase (*DynShare_CPI*) produces worst average BSLD compared to the *DynShare_mixed* case. This is due to the fact that the historical traces are mostly composed by CPU-intensive jobs with low CPI values and therefore the impact of power reduction is heavier. Then, compared to the synthetic benchmarks case we observe a relative decrease of the performance of *LS* & *DEC* w.r.t. *DynShare_mixed* (and other methods able to modulate the power).

The discrepancy of results between historical and synthetic traces needs to be explained. The key difference between the two types of benchmarks is the job arrival frequency, which is much higher in the synthetic benchmarks. For example if we look at the case of instances of 100 jobs, while time windows of synthetic benchmarks range from 5 minutes to 2 hours the average time window of historical traces is more or less 6 hours. These characteristics of the historical workload are probably due to the fact that Eurora has been used as a prototype and it was never heavily loaded in the measurement period, while a machine in production would be much more loaded on average. Another key difference is that many jobs in the historical traces have extremely short durations - around 15%-20% of jobs last less than 1 minute; in the synthetic benchmarks there are no such short jobs. Drastically decreasing the job arrival frequency changes the difficulty of the dispatching problem because fewer jobs are in the system, fewer jobs are forced to wait due to unavailable resources and using a "smart" scheduling policy loses its benefits. Moreover, the power aspect acquires greater relevance and the methods able to modulate the power consumption become preferable.

The correlation between the performance difference of the proactive methods (*LS* & *DEC*) w.r.t. methods using RAPL and the job arrival time window can be easily seen in Figure 6, which illustrates the performance difference between *DEC* and *DynShare_mixed* with varying time windows. On the



(a) 100 Jobs



(b) 400 Jobs

Figure 5: Average BSLD; historical traces

x-axis the job arrival time window is displayed and the *y*-axis shows the BLSD percentage difference, for each power cap[9], between the considered methods; negative values correspond to the cases when *DEC* outperforms *DynShare_mixed* and vice-versa. The figure presents the results for 4 different power cap values and for the average results computed on all power cap (yellow square markers). The arrival times follow a random uniform distribution. It can be easily observed that if the window size increases (thus the job arrival frequency decreases) *DynShare_mixed* begins to outperform *DEC* if the power cap is smaller than 50%; even with larger power budget the relative performance of *DEC* degrades with the increase in time window size - experiments performed with even larger time windows, not shown in this graph, confirm the trend observed here. This correlation gave us an insight to combine the benefits of proactive and power-modulating methods in order to obtain a dispatcher well suited to cope with any type of workload. The main idea is that if the job arrival frequency is high *LS* & *DEC* provide better results when job arrival frequency decreases under a threshold the best method is some variant of *DynShare*. We therefore implemented the new approach *DEC+RAPL* that

[9] $\%DIF^{pcap} = 100 \times \frac{BSLD^{pcap}_{DEC} - BSLD^{pcap}_{RAPL\_MIX}}{BSLD^{pcap}_{RAPL\_MIX}}$

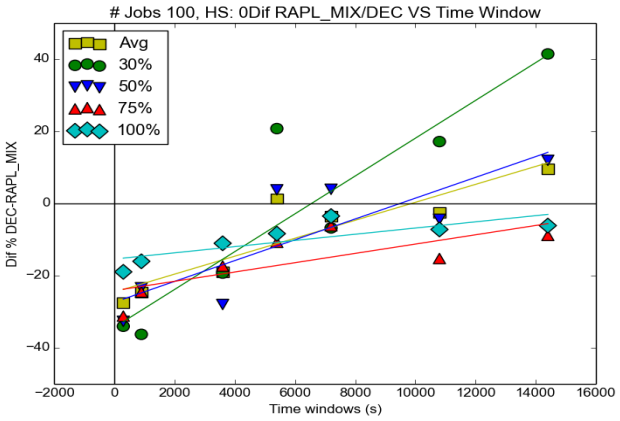


Figure 6: Average BSLD; 100 jobs from historical traces

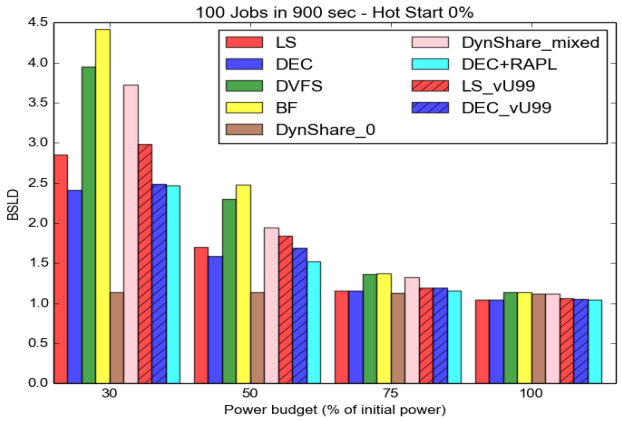


Figure 7: Average BSLD; 100 jobs in 900s; HS=0%; uniform distribution

uses both *DEC* and *DynShare_mixed*, depending on the job arrival frequency. The correct threshold was obtained through empirical evaluation and it is equal to one job every two minutes. The experiments prove that combining the best of both world yields the best results. Looking again at Fig. 5a and Fig. 5b we focus now on the last column on the right which represents *DEC+RAPL*. The new dispatcher obtains better or equal results than the other methods in almost all situation (except the case of 30% power budget and 200 jobs). Even tough not shown here, the results of the experiments conducted on the synthetic benchmarks are analogous. For example, with an arrival window of 5 minutes (such as the one of previous figures) *DEC+RAPL* has the same performance of *DEC* because with such a high arrivals frequency the RAPL component is never activated.

5.2.4. Mispredictions Impact

As noted in Section 4.1 the power consumption predictions can be inaccurate. This is a problem especially in the case of under-prediction, i.e. the forecast power consumption is smaller than the real one, because *LS* & *DEC* might produce schedules violating the power budget (they enforce a constraint based on the predictions). A possible solution is to impose them a tighter power cap than the target one, so that if under-predictions happen the desired budget would still be respected; however, decreasing the power cap has the downside of performance degradation. We conducted a set of additional experiments to quantify the performance decrease due to handling the under-predictions for our proactive dispatchers, *LS* & *DEC*.

In the new batch of experiments we use the previous jobs instances but we impose that 10% of the job in an instance are under-predicted; for these jobs the predicted power is 40% smaller than the real one. In order to provide a robust analysis we are forcing an under-prediction rate more severe than the one observed when testing our prediction models in [55]. We then run again *LS* & *DEC* (RAPL-based methods use the real power thus do not require additional experiments) using a smaller power budget, i.e. $power\_cap = 0.95 \cdot power\_target$. After a preliminary analysis we discovered that an extremely small power cap decrease is sufficient to ensure that the target power cap is respected; for example, using a power cap 1% smaller than the target cause both *LS* & *DEC* not to violate the budget.

In Figure 7 we see the results of the experiments with under-predictions in the case of 100 jobs (synthetic benchmark). We see the main dispatchers shown before plus *LS* & *DEC* using a power cap value 1% smaller than the target, represented by the hatched columns *LS_vU99* & *DEC_vU99*. The target power budget is respected while the BSLD slightly rises; taking into account all the instances, given a 1% power cap reduction w.r.t. target, the average BSLD increases by 4.2% for *LS* and by 4.4% for *DEC*. For a more conservative reduction of 5% the BSLD increases by around 6%-7%.

6. Conclusion

In this paper we have dealt with the problem of limiting the power consumption of a HPC system. We developed a job dispatcher to enforce such power constraint acting on the job execution order alone. We proposed two different approaches: 1) a heuristic algorithm (inspired by the class of Priority-Rules-Based algorithm) and 2) a hybrid approach that decomposes the dispatching problem in its two components, the scheduling phase (solved using Constraint Programming) and the allocation phase (solved through a heuristic technique). The heuristic method is quicker but the hybrid approach can provide the best solutions - while still amply respecting real-time execution constraints.

A fundamental aspect of our approach is the requirement to know the power consumption of a job before its real execution. This estimate is used to dispatch all jobs while guaranteeing that the power budget will never be violated. Therefore we proposed a predictive model (created using Machine Learning techniques) capable of estimating the power consumption of a job with high accuracy, using only the information available at dispatching-time. Given that high accuracy does not imply perfect prediction, we can guarantee that our dispatcher will enforce a power cap only in a confidence interval specified by the quality of the prediction.

We then implemented several alternative power capping methods taken from the literature and we compared their performance against the one obtained with our methods. We run experiments with different level of power budgets, instances of different sizes and different initial conditions. The results show that our methods (particularly the hybrid one) perform better than the vast majority of remaining methods in most of the situations. This is especially true if we consider problems where the power constraint is not extremely tight, due to the proactive nature of our approach (we generate optimal plans considering all jobs in the system queue). Even in the worst situations, for example with systems in a non-empty initial state, our dispatchers offer performance comparable to the one provided by the methods from the literature. We also proved that our methods have a very good scalability, continuing to outperform the competition even with larger instances.

The experiments performed with historical traces revealed the impact of the workload characteristics (i.e. the job arrival frequency) on the quality of the solutions found by our methods. Therefore we also proposed an additional dispatcher that mixes the smart planning of our job dispatchers and the ability of modulating the power consumption at run time typical of the best techniques from the state-of-the-art. This method combines the strengths of both approaches and manage to outperforms all remaining dispatchers. As noted before, a crucial component of our approach is the accuracy of the power prediction, with higher accuracy leading to better performance. We also took into account the robustness of our method to inaccurate predictions; we studied this aspect by using varying level of power caps to cope with possible mispredictions. The results reveal that using a tighter power bound than the target one guarantees the robustness of the approach without decreasing significantly the overall performance.

We have identified some future research directions. We want to test our approach on a real environment and implement it on a real production machine. We will apply our power prediction methodology to other HPC systems to test its accuracy on different workloads; for this purpose we are working in strict collaboration with Cineca consortium to obtain power measurements from other Top500 and Green500 supercomputers. Finally, in this paper we only considered rigid jobs in terms of resources while nowadays an emerging trend points towards the adoption of moldable (and/or malleable) jobs; including this class of workload in our dispatcher is a very promising area we are keen on exploring.

Acknowledgements

This work was partially supported by the FP7 ERC Advanced project MULTITHERMAN (g.a. 291125). We also want to thank CINECA and Eurotech for granting us the access to their systems.

[1] P. Kogge, D. R. Resnick, Yearly update: exascale projections for 2013., 2013. doi:10.2172/1104707.

[2] J. J. Dongarra, H. W. Meuer, E. Strohmaier, 29th top500 Supercomputer Sites, Tech. rep., Top500.org (Nov. 1994).

[3] H. Fu, J. Liao, J. Yang, et Al., The sunway taihulight supercomputer: system and applications, Science China Information Sciences 59 (7).

[4] K. Bergman, S. Borkar, D. Campbell, et al., Exascale computing study: Technology challenges in achieving exascale systems (September 2008).

[5] C. Lefurgy, X. Wang, M. Ware, Power capping: a prelude to power shifting, Cluster Computing 11 (2).

[6] T. Bridi, A. Bartolini, M. Lombardi, et Al., A constraint programming scheduler for heterogeneous high-performance computing machines, IEEE Transactions on Parallel and Distributed Systems 27 (10) (2016) 2781–2794. doi:10.1109/TPDS.2016.2516997.

[7] A. Bartolini, A. Borghesi, T. Bridi, et Al., Proactive workload dispatching on the EURORA supercomputer, in: Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings, 2014.

[8] M. Etinski, J. Corbalan, J. Labarta, M. Valero, Understanding the future of energy-performance trade-off via {DVFS} in {HPC} environments, Journal of Parallel and Distributed Computing 72 (4).

[9] C.-Y. Lee, T.-Y. Lin, R.-G. Chang, Power-aware code scheduling assisted with power gating and dvs, Future Generation Computer Systems 34 (2014) 66–75.

[10] J. Wu, Energy-efficient scheduling of real-time tasks with shared resources, Future Generation Computer Systems 56 (2016) 179–191.

[11] S. K. Tesfatsion, E. Wadbro, J. Tordsson, A combined frequency scaling and application elasticity approach for energy-efficient cloud computing, Sustainable Computing: Informatics and Systems 4 (4) (2014) 205–214.

[12] K. De Vogeleer, G. Memmi, P. Jouvelot, Parameter sensitivity analysis of the energy/frequency convexity rule for application processors, Sustainable Computing: Informatics and Systems 15 (2017) 16–27.

[13] G. Varsamopoulos, S. K. Gupta, Energy proportionality and the future: Metrics and directions, in: Parallel Processing Workshops (ICPPW), 2010 39th International Conference on, IEEE, 2010.

[14] T. Patki, D. K. Lowenthal, B. Rountree, et Al., Exploring hardware overprovisioning in power-constrained, high performance computing, in: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing, ICS '13, ACM, New York, NY, USA, 2013.

[15] J. Hikita, A. Hirano, H. Nakashima, Saving 200kw and $200 k/year by power-aware job/machine scheduling, in: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, 2008, pp. 1–8. doi:10.1109/IPDPS.2008.4536218.

[16] H. Shoukourian, T. Wilde, A. Auweter, A. Bode, Power variation aware configuration adviser for scalable hpc schedulers, in: High Performance Computing Simulation (HPCS), 2015 International Conference on, 2015.

[17] J. Meng, S. McCauley, F. Kaplan, V. J. Leung, A. K. Coskun, Simulation and optimization of hpc job allocation for jointly reducing communication and cooling costs, Sustainable Computing: Informatics and Systems 6 (2015) 48–57.

[18] H. David, E. Gorbatov, U. R. Hanebutte, et Al., Rapl: Memory power estimation and capping, in: Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED '10, ACM, New York, NY, USA, 2010. doi:10.1145/1840845.1840883.

[19] O. Sarood, A. Langer, A. Gupta, L. Kale, Maximizing throughput of overprovisioned hpc data centers under a strict power budget, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '14, 2014.

[20] O. Mämmelä, M. Majanen, R. Basmadjian, et Al., Energy-aware job scheduler for high-performance computing, Computer Science-Research and Development 27 (4) (2012) 265–275.

[21] P. E. Bailey, D. K. Lowenthal, V. Ravi, et Al., Adaptive configuration selection for power-constrained heterogeneous systems, in: Proceedings of the 2014 Brazilian Conference on Intelligent Systems, BRACIS '14, IEEE Computer Society, Washington, DC, USA, 2014.

[22] T. Patki, D. K. Lowenthal, A. Sasidharan, et Al., Practical resource management in power-constrained, high performance computing, in: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, HPDC '15, ACM, New York, NY, USA, 2015, pp. 121–132. doi:10.1145/2749246.2749262.

[23] A. W. Mu'alem, D. G. Feitelson, Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling, IEEE Trans. Parallel Distrib. Syst. 12 (6) (2001) 529–543. doi:10.1109/71.932708.

[24] Y. Inadomi, T. Patki, K. Inoue, et Al, Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing, in: Proceedings of the International Conference for High Performance

Computing, Networking, Storage and Analysis, SC '15, ACM, New York, NY, USA, 2015, pp. 78:1–78:12. doi:10.1145/2807591.2807638.

[25] V. W. Freeh, D. K. Lowenthal, F. Pan, et Al., Analyzing the energy-time trade-off in high-performance computing applications, IEEE Trans. Parallel Distrib. Syst. 18 (6). doi:10.1109/TPDS.2007.1026.

[26] C. Hsu, W. Feng, A power-aware run-time system for high-performance computing, in: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, IEEE Computer Society, 2005.

[27] M. Etinski, J. Corbalan, J. Labarta, M. Valero, Optimizing job performance under a given power constraint in hpc centers, in: Green Computing Conference, 2010 International, 2010. doi:10.1109/GREENCOMP.2010.5598303.

[28] M. Etinski, J. Corbalan, J. Labarta, et Al., Utilization driven power-aware parallel job scheduling, Computer Science - Research and Development 25 (3) (2010) 207–216.

[29] M. Etinski, J. Corbalan, J. Labarta, M. Valero, Parallel job scheduling for power constrained {HPC} systems, Parallel Computing 38 (12). doi:http://dx.doi.org/10.1016/j.parco.2012.08.001.

[30] N. Kumar, D. P. Vidyarthi, An energy aware cost effective scheduling framework for heterogeneous cluster system, Future Generation Computer Systems.

[31] D. Bodas, J. Song, M. Rajappa, A. Hoffman, Simple power-aware scheduler to limit power consumption by hpc system within a budget, in: Proceedings of the 2Nd International Workshop on Energy Efficient Supercomputing, E2SC '14, IEEE Press, Piscataway, NJ, USA, 2014, pp. 21–30. doi:10.1109/E2SC.2014.8.

[32] D. A. Ellsworth, A. D. Malony, B. Rountree, M. Schulz, Dynamic power sharing for higher job throughput, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, ACM, New York, NY, USA, 2015, pp. 80:1–80:11. doi:10.1145/2807591.2807643.

[33] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, S. Sankar, The need for speed and stability in data center power capping, Sustainable Computing: Informatics and Systems 3 (3) (2013) 183–193.

[34] B. Khemka, R. Friese, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, R. Rambharos, S. Poole, Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system, Sustainable Computing: Informatics and Systems 5 (2015) 14–30.

[35] K. Leal, Energy efficient scheduling strategies in federated grids, Sustainable Computing: Informatics and Systems 9 (2016) 33–41.

[36] A. Kaushik, D. P. Vidyarthi, A green energy model for resource allocation in computational grid using dynamic threshold and ga, Sustainable Computing: Informatics and Systems 9 (2016) 42–56.

[37] K. Li, Energy-efficient task scheduling on multiple heterogeneous computers: Algorithms, analysis, and performance evaluation, IEEE Transactions on Sustainable Computing 1 (1) (2016) 7–19.

[38] S. Pakin, C. Storlie, M. Lang, et Al., Power usage of production supercomputers and production workloads, Concurr. Comput. : Pract. Exper. 28 (2) (2016) 274–290. doi:10.1002/cpe.3191.

[39] C. Storlie, J. Sexton, S. Pakin, et Al., Modeling and predicting power consumption of high performance computing jobs, arXiv preprint arXiv:1412.5247.

[40] H. Huang, M. Fan, G. Quan, Thermal aware overall energy minimization scheduling for hard real-time systems, Sustainable Computing: Informatics and Systems 3 (4) (2013) 274–285.

[41] J. Choi, S. Govindan, B. Urgaonkar, et Al., Profiling, prediction, and capping of power consumption in consolidated environments, in: Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on, IEEE, 2008.

[42] G. Chetsa, L. Lefevre, J. Pierson, et Al., Exploiting performance counters to predict and improve energy performance of hpc systems, Future Generation Computer Systems 36.

[43] G. Contreras, M. Martonosi, Power prediction for intel xscale®processors using performance monitoring unit events, in: Proceedings of the 2005 International Symposium on Low Power Electronics and Design, ISLPED '05, ACM, New York, NY, USA, 2005, pp. 221–226. doi:10.1145/1077603.1077657.

[44] M. Jarus, A. Oleksiak, T. Piontek, J. Weglarz, Runtime power usage estimation of hpc servers for various classes of real-life applications, Future Generation Computer Systems 36 (2014) 299–310.

[45] H. Shoukourian, T. Wilde, A. Auweter, A. Bode, D. Tafani, Predicting the energy and power consumption of strong and weak scaling hpc applications, Supercomputing frontiers and innovations 1 (2) (2014) 20–41.

[46] A. Auweter, A. Bode, M. Brehm, et Al., A case study of energy aware scheduling on supermuc, in: J. Kunkel, T. Ludwig, H. Meuer (Eds.), Supercomputing, Vol. 8488 of Lecture Notes in Computer Science, Springer International Publishing, 2014.

[47] Eurora page on the cineca web site, http://www.cineca.it/en/content/eurora.

[48] Prace. partnership for advanced computing in europe.

[49] Cineca inter-university consortium web site, http://www.cineca.it//en, accessed: 2014-04-14.

[50] A. Bartolini, M. Cacciari, C. Cavazzoni, et Al., Unveiling eurora - thermal and power characterization of the most energy-efficient supercomputer in the world, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2014, 2014.

[51] F. Rossi, P. Van Beek, T. Walsh, Handbook of constraint programming, Elsevier, 2006.

[52] P. Baptiste, C. L. Pape, W. Nuijten, Constraint-based scheduling, Kluwer Academic Publishers, 2001.

[53] A. Borghesi, C. Conficoni, M. Lombardi, et Al., MS3: A mediterranean-stile job scheduler for supercomputers - do less when it's too hot!, in: 2015 International Conference on High Performance Computing & Simulation, HPCS 2015, Amsterdam, Netherlands, July 20-24, 2015, 2015. doi:10.1109/HPCSim.2015.7237025.

[54] A. Borghesi, F. Collina, M. Lombardi, et Al., Power capping in high performance computing systems, in: Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings, 2015.

[55] A. Borghesi, A. Bartolini, M. Lombardi, et Al., Predictive Modeling for Job Power Consumption in HPC Systems, Springer International Publishing, Cham, 2016, pp. 181–199.

[56] J. R. Quinlan, Induction of decision trees, Mach. Learn. 1 (1) (1986) 81–106. doi:10.1023/A:1022643204877.

[57] L. Breiman, Random forests, Mach. Learn. 45 (1). doi:10.1023/A:1010933404324.

[58] R. Haupt, A survey of priority rule-based scheduling, Operations-Research-Spektrum 11 (1). doi:10.1007/BF01721162.

[59] P. Baptiste, P. Laborie, C. Le Pape, W. Nuijten, Constraint-based scheduling and planning, Foundations of Artificial Intelligence 2 (2006) 761–799.

[60] S. Wallace, X. Yang, V. Vishwanath, et Al., A data driven scheduling approach for power management on hpc systems, in: Proceedings of SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, SC, Vol. 16, 2016.

[61] A. P. Marathe, Evaluation and optimization of turnaround time and cost of hpc applications on the cloud.

[62] A. Bartolini, M. Cacciari, A. Tilli, L. Benini, M. Gries, A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores, in: Proceedings of the 20th symposium on Great lakes symposium on VLSI, ACM, 2010, pp. 311–316.

[63] F. Fraternali, A. Bartolini, C. Cavazzoni, L. Benini, Quantifying the impact of variability and heterogeneity on the energy efficiency for a next-generation ultra-green supercomputer, IEEE Transactions on Parallel and Distributed Systems.

[64] M. D. De Assunção, R. Buyya, Performance analysis of multiple site resource provisioning: Effects of the precision of availability information, in: Proceedings of the 15th International Conference on High Performance Computing, HiPC'08, Springer-Verlag, Berlin, Heidelberg, 2008.

[65] M. Stillwell, F. Vivien, H. Casanova, Dynamic fractional resource scheduling for hpc workloads, in: Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on, IEEE, 2010, pp. 1–12.

[66] E. Gaussier, D. Glesser, V. Reis, D. 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, ACM, 2015, p. 64.