Power capping in high performance computing systems

Power Capping in High Performance Computing Systems

Andrea Borghesi, Francesca Collina, Michele Lombardi, Michela Milano, Luca Benini
DISI/DEI, University of Bologna

Abstract

Power consumption is a key factor in modern ICT infrastructure, especially in the expanding world of High Performance Computing, Cloud Computing and Big Data. Such consumption is bound to become an even greater issue as supercomputers are envisioned to enter the Exascale by 2020, granted that they obtain an order of magnitude energy efficiency gain. An important component in many strategies devised to decrease energy usage is "power capping", i.e., the possibility to constrain the system power consumption within certain power budget. In this paper we propose two novel approaches for power capped workload dispatching and we demonstrate them on a real-life high-performance machine: the Eurora supercomputer hosted at CINECA computing center in Bologna. Power capping is a feature not included in the commercial Portable Batch System (PBS) dispatcher currently in use on Eurora. The first method is based on a heuristic technique while the second one relies on a hybrid strategy which combines a CP and a heuristic approach. Both systems are evaluated and compared on simulated job traces.

1 Introduction

Supercomputer peak performance is expected to reach the ExaFLOP level in 2018-2020 [14], however energy efficiency is a key challenge to be addressed to reach this milestone. Today's most powerful Supercomputer is Tianhe-2 which reaches 33.2 PetaFlops with 17.8 MWatts of power dissipation [13]. Exascale supercomputers built upon today's technology would led to an unsustainable power demand (hundreds of MWatts) while according to [9] an acceptable range for an Exascale supercomputer is 20MWatts; for this goal, current supercomputer systems must obtain significantly higher energy efficiency, with a limit of 50GFlops/W. Today's most efficient supercomputer achieves 5.2 GFlops/W, thus we still need to close an order of magnitude gap to fulfill the Exascale requirements.

Almost all the power consumed by HPC systems is converted into heat. In addition to the power strictly needed for the computation - which measures only the computational efficiency - the cooling infrastructure must be taken

into account, with its additional power consumption. The extra infrastructure needed for cooling down the HPC systems has been proved to be a decisively limiting factor for the energy performance [3]; a common approach taken to address this problem is the shift from air cooling to the more efficient liquid cooling [10].

Hardware heterogeneity as well as dynamic power management have started to be investigated to reduce the energy consumption [16][2]. These low-power techniques have been derived from the embedded system domain where they have proven their effectiveness [1]. However, a supercomputer is different from a mobile handset or a desktop machine. It has a different scale, it cannot be decoupled by the cooling infrastructures and its usage mode is peculiar: it is composed by a set of scientific computing applications which run on different datasets with a predicted end time [6]. Finally supercomputers are expensive (6 orders of magnitude more than an embedded device [21]) making it impossible for researchers to have them on their desk. These features have limited the development of ad-hoc power management solutions.

Current supercomputers cooling infrastructures are designed to withstand power consumption at the peak performance point. However, the typical supercomputer workload is far below the 100% resource utilization and also the jobs submitted by different users are subject to different computational requirements[29]. Hence, cooling infrastructures are often over-designed. To reduce overheads induced by cooling over-provisioning several works suggest to optimize job dispatching (resource allocation plus scheduling) exploiting non-uniformity in thermal and power evolutions [8][19][20][23]. Currently most of these works are based on simulations and model assumptions which unfortunately are not mature enough to be implemented on HPC systems in production, yet.

With the goal of increasing the energy efficiency, modern supercomputers adopt complex and hybrid cooling solutions which try to limit the active cooling (chiller, air conditioner) by allowing direct heat exchange with the ambient (Free-cooling). Authors in [12] show for the 2013's top GEEN500 supercomputer that the cooling costs increase four times when the ambient temperature moves from 10°C to 40°C. Moreover the authors show that for a given ambient temperature there is a well-defined maximum power budget which guarantees efficient cooling. With an ambient temperature of 10°C the power budget which maximizes the efficiency is 45KWatt while at 40°C is 15KWatt. Unfortunately, today's ITs infrastructure can control its power consumption only reducing the power and performance of each computing node. This approach is not suitable for HPC system where users require to execute their application in a portion of the machine with guaranteed performance. A solution commonly adopted in HPC systems is *power capping*[25][17][15][27][22], which means forcing a supercomputer not to consume more than a certain amount of power at any given time. In this paper we study a technique to achieve a power capping by acting on the number of job entering the system.

We propose two different methods to enforce power capping constraints on a real supercomputer: 1) a Priority Rules Based algorithm and 2) a novel hybrid approach which combines a CP and a heuristic technique.

2 System Description and Motivations for Using CP

The Eurora Supercomputer: As described in [7] 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 MIC accelerators. Every node has 16GB of installed RAM memory. A few nodes (external to the rack) allow the interaction between Eurora and the outside world, in particular a login node connects Eurora to the users and runs the job dispatcher (PBS). A key element of the energy efficiency of the supercomputer is a hot liquid cooling system, i.e. the water inside the system can reach up to 50°C. This obviously allows to save a lot of energy, since no power is used to actively cool down the water; furthermore the heat in excess can be reused as an energy source for other activities.

The PBS Dispatcher: The tool currently used to manage the workload on Eurora system is PBS [28] (Portable Batch System), a proprietary job scheduler by Altair PBS Works which has the task of allocating computational activities, i.e. batch jobs, among available computing resources. The main components of PBS are a server (which manages the jobs) and several daemons running on the execution hosts (i.e. the 64 nodes of Eurora), which track the resource usage and answer to polling requests about the host state issued by the server component.

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 MICs per node (never both of them at the same time); 4) the amount of required memory per node; 5) the maximum execution time. All processes that exceed their maximum execution time are killed. The main available queues on the Eurora system are called *debug*, *parallel*, and *longpar*. PBS periodically selects a job to be executed considering the current state of the nodes - trying to find enough available resources to start the job. If there are not enough available resources the job is returned to its queue and PBS considers the following candidate. The choices are guided by priority values and hard-coded constraints defined by the Eurora administrators with the aim to have a good machine utilization and small waiting times. For more detailed information regarding the system see [6].

Why CP? In its current state, the PBS system works mostly as an on-line heuristic, incurring the risk to make poor resource assignments due to the lack of an overall plan; furthermore, it does not include a power capping feature yet. Other than that, as we shown in our previous work the lack of an overall plan may lead to poor resource assignments. The task of obtaining a dispatching

plan on Eurora can be naturally framed as a resource allocation and scheduling problem, for which CP has a long track of success stories. Nevertheless since the problem is very complex and we are constrained by very strict time limits (due to the real-time requirements of a supercomputer dispatcher) it is simply not possible to always find a optimal solutions. Therefore, we used CP in combination with multiple heuristic approaches in order to quickly find the resource allocation and schedule needed.

3 Problem Definition

We give now a more detailed definition of the problem. Each job i enters the system at a certain arrival time q_i, by being submitted to a specific queue (depending on the user choices and on the job characteristics). By analyzing existing execution traces coming from PBS, we have determined an estimated waiting time for each queue, which applies to each job it contains: we refer to this value as ewt_i.

When submitting the job, the user has to specify several pieces of information, including the maximum allowed execution time D_i, the maximum number of nodes to be used rn_i, and the required resources (cores, memory, GPUs, MICs). By convention, the PBS system considers each job as if it was divided into a set of exactly rn_i identical "job units", to be mapped each on a single node. Formally, let R be a set of indexes corresponding to the resource types (cores, memory, GPUs, MICs), and let the capacity of a node $k \in K$ for resource $r \in R$ be denoted as $cap_{k,r}$. Let $rq_{i,r}$ be the requirement of a unit of job i for resource r. The dispatching problem at time τ requires to assign a start time $s_i \geq \tau$ 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 $s_i = \tau$ 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.

Along with the aforementioned resources in this problem we introduce an additional finite resource, the power. This will allow us to model and enforce the power capping constraint. The power capping level, i.e. the maximal amount of power consumed by the system at any given time, is specified by the user and represents the capacity of the fake power resource; the sum of all the power consumed by the running job plus the sum of power values related to idle resources must never exceed the given limit throughout the execution of the whole schedule. Another important thing to notice is that the power "resource" has not a linear behaviour in terms of the computational workload: the first activation of a core in a node causes greater power consumptions for the remaining cores in that node.

The goal is to reduce the waiting times, as a measure of the Quality of Service guaranteed to the supercomputer users.

4 The PRB approach

First, we implemented a dispatcher which belongs to a class of scheduling techniques known in the literature as *Priority Rules Based* scheduling (PRB)[18]. The main idea underlying this approach is to order a set of tasks which need to be scheduled, constructing the ordered list by assigning priority for each task. Tasks are selected in the order of their priorities and each selected task is assigned to a node; even the resource are ordered and the ones with higher priority are preferred - if available. This is an heuristic technique and it is obviously not able to guarantee an optimal solution but has the great advantage of being extremely fast.

The jobs are ordered w.r.t to their expected wait 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[1]).

The PRB algorithm proceeds 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 will try 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.

Algorithm 1 shows the pseudo code of the PRB algorithm. Lines 1-6 initialize the algorithm; J is the set of activities to be scheduled and R is the set of nodes (ordered with the $orderByRules()$ function which encapsulates the priority rules). Then the algorithm proceeds while there are still jobs to be scheduled; at every iteration we try to start as many jobs as possible (line 8). Each job unit is considered (line 10) and the availability of resources on every node is taken into account; the function $checkAvailability(rn_i, r)$ (line 12) returns true if there are enough available resources on node r to map unit rn_i. If it is possible the unit is then mapped and the system usage is updated ($updateUsages(rn_i, R)$, line 13), vice versa we register that at least a job unit could not be mapped (line 16). If all the units of a job have been mapped (line 17-21), then the job can actually start and is removed from the pool of

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

Algorithm 1: PRB algorithm

```
1  time ← 0;
2  startTimes ⟵ ∅;
3  endTimes ⟵ ∅;
4  runningJobs ⟵ ∅;
5  orderByRules(R);
6  orderByRules(J);
7  while J ≠ ∅ do
8      for j ∈ J do
9          canBeMapped ← true;
10         for rn_i ∈ j do
11             for r ∈ R do
12                 if checkAvailability(rn_i, r) then
13                     updateUsages(rn_i, R);
14                     break;
15                 else
16                     canBeMapped ← false
17         if canBeMapped = true then
18             J ⟵ J − {j};
19             runningJobs = runningJobs ∪ {j};
20             startTimes(j) ← time;
21             endTimes(j) ← time + duration(j);
22         else
23             undoUpdates(j, R);
24     orderByRules(R);
25     orderByRules(J);
26     time ← min(endTimes);
```

jobs that still need to be scheduled. Conversely, if the job cannot start we must undo the possible changes made to the system usage we made in advance ($updateUsages(rn_i, R)$, line 23). Finally, after having dealt with all schedulable jobs we reorder the activities and nodes (the activity pool is changed and the nodes order depends on the usages), lines 24-25, and compute the closest time point where some used resource becomes free, following time-event, i.e. the minimal end time of the running activities (line 26).

It is important to note that since in this problem we have no deadline on the single activities nor a constraint on the global makespan, 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.

5 Hybrid Approach

As previously discussed in [6] the task of obtaining a proactive dispatching plan on Eurora can be naturally framed as a resource allocation and scheduling problem. Constraint Programming (CP) techniques have shown great success when dealing with this kind of problem, thanks to the expressive and flexible language used to model the problem and powerful algorithms to quickly find good quality solutions[5]. In this Section we describe the mixed approach we used to solve the resource allocation and scheduling problem under power capping constraints.

The key idea of our method is to decompose the allocation and scheduling 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. This two stages are repeated n times, where n has 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. 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.

In this work 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 additional resource with its own capacity (i.e. the user-specified power cap), which we cannot "over-consume" at any moment[11]. In this way, the power used in the active nodes (i.e. those on which a job is running) summed to the power consumed by the idle nodes will never exceed the given threshold.

The next sections will describe in more detail the two stages of the decomposed approach.

5.1 Scheduling Problem

The scheduling problem consists in deciding the start times of a set of activities $i \in I$ 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 resource with a capacity Cap_r^T which is the sum of all the single resource capacities, $Cap_r^T = \sum_{k \in K} cap_{k,r} \quad \forall r \in R$. As mentioned before the power is considered as an additional 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 employs other relaxations: 1) only the power required by running jobs (active power) is considered, i.e. we use a lower bound of the total power consumed in the system, 2) we assume that the jobs always run on the nodes which require less power and 3) we overlook the non-linear behaviour of the power consumption. These relaxations may produce infeasible solutions, since a feasible schedule must take into account that the resource are actually split among heterogeneous nodes and consider also the idle nodes for the power capping. The following stage of our method will naturally fix these possible infeasibilities during the allocation phase.

We define the scheduling model using Conditional Intervals Variables (CVI)[24]. A CVI τ represents an interval of time: the start of the interval is referred to as $s(\tau)$ and its end as $e(\tau)$; the duration is $d(\tau)$. The interval may or may not be present, depending on the value of its existence expression $x(\tau)$ (if not present it does not affect the model). CVIs can be subject to several different constraints, among them the `cumulative` constraint[4] to model finite capacity resources.

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

where τ is the vector with all the interval vars, where req_r are the job requirements for resource r - using past execution traces we learned a simple model to estimate the power consumed by a job based on its requirements. As mentioned in section 3 this model is not linear. 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*, 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):

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

To solve the scheduling model we implemented a custom search strategy derived from the Schedule Or Postpone strategy [26]. The criteria used to select an activity among all the available ones at each decision node follows the priority rules used in the heuristic algorithm, thus preferring jobs that can start first and whose resource demand is lower. This strategy proved to be very effective and able to rapidly find good solutions w.r.t. the objective function we are considering in this problem. With different goals we should change the search strategy accordingly (as with the priority rules).

5.2 Allocation Problem

The allocation problem consists in mapping each job unit on a node. Furthermore, in our approach the allocation stage is also in charge of fixing the

infeasibilities generated at the previous stage. In order to solve this problem we developed an algorithm which falls in the PRB category. Typical PRB schedulers decide both mapping and start times, whereas in our hybrid approach we need only to allocate the jobs.

The behaviour of this algorithm (also referred as *mapper*) is very close to the PRB one described in Section 4 and in particular the rules used to order jobs and resources are identical. The key difference is that now we already know the start and end times of the activities (at least the possible ones, they may change if any infeasibility is detected). This algorithm proceeds by time-step: at each time event t it considers only the jobs that should start at time t according to the relaxed CP model described previously, while the simple PRB algorithm considers at each time-event all the activities that still need to be scheduled.

During this phase the power is also taken into account, again seen as a finite resource with capacity defined by the power cap; here we consider both active and idle nodes powers. If the job can be mapped somewhere in the system the start time t from the previous stage is kept, otherwise - if there are not enough resources available to satisfy the requirements - the previously computed start time is discarded and the job will become eligible to be scheduled at the next time-step $t^{'}$. At the next time event $t^{'}$ all the jobs that should start are considered, plus the jobs that possibly have been postponed due to scarce resources at the previous time-step. Through this postponing we are fixing the infeasibilities inherited from the relaxed CP model.

Again, since in this problem we have no constraints on the total duration of a schedule, it is always possible to delay a job until the system will have enough available resources to run it, thus this method is guaranteed to find a feasible solution.

5.3 Interaction between the stages

We designed a basic interaction mechanism between the two stages with the goal to find better quality solutions. The main idea is to exploit the information regarding the infeasibilities found during the allocation phase to lead the search of new solutions for the relaxed scheduling problems. In particular whenever we detect a resource over-usage at time *tau* which requires a job to be postponed during the second stage of our model we know that the set of job running at time τ is a *Conflict Set* (not minimal), i.e. not all activities in the set can run concurrently. A possible solution for a conflict set is for example to introduce precedence relationships among activities (thus eliminating jobs from the conflict set) until the infeasibility is resolved.

In our approach we use the conflict set detected in the mapping phase to generate a new set of constraints which impose that not all jobs in the set will run at the same time. We then introduce in the CP model a fake cumulative constraint for each conflict set. The jobs included in such cumulative constraint are those included in the conflict set, each of them with a "resource" demand of one; the capacity not to be exceeded is given by the conflict set size minus one. These cumulative constraints enforce that the jobs involved will not run at

the same time. This mechanism does not guarantee yet that the new solution found by the CP scheduler will be feasible since the conflict sets we detect are not minimal, nevertheless it provides a way to help the CP model to produce solutions which will require less "fixing" by the mapper.

In conjunction with the additional cumulative constraint at each iteration we also cast a further constraint on the objective variable in order to force the new solution to improve in comparison to the previous one.

6 Added Value of CP

The dispatcher we realized is currently a prototype: it will eventually be deployed on the Eurora supercomputer, but this requires still considerable development and research effort (at the same time a previous version without power capping of this model[6] has already successfully been implemented). At this stage we are interested in investigating the kind of impact that introducing a power capping feature may have on the dispatcher behaviour and performance.

The dispatcher we realized can work in two different modes: *off-line*, i.e. the resource allocation and the schedule are computed before the actual execution, and *on-line*, i.e. allocation and scheduling decisions are taken at run-time, upon the arrival of new tasks in the system. Clearly the actual implementation on the supercomputer would require the on-line strategy since the workload is submitted by users and not statically decided a priori. At the same time we decided to use the off-line strategy to perform the experiments described in this paper since we wanted to test our approaches with reproducible conditions. We also needed our techniques to be stable in order to implement them on a production system and the off-line strategy allows us to better verify that.

The proposed methods were implemented using or-tools[2], Google's software suite for combinatorial optimization. We performed an evaluation of all our approaches on PBS execution traces collected from Eurora in a timespan of several months. From the whole set of traces we extracted different batches of jobs submitted at various times and we used them to generate several job instances of different size, i.e. the number of jobs per instance (a couple of hundreds of instances for each size). Since in this experimental evaluation we were concerned only in the off-line approach the real enter queue times were disregarded and in our instances we assume that all jobs enter the system at the same time. The main performance metric considered is the time spent by the jobs in the queues while waiting their execution to begin (ideally as low as possible).

6.1 Evaluation of Our Models

We decided to make our experiments using two artificial versions of the Eurora machine: A) a machine composed with 8 nodes and B) a machine with 32 nodes. In addition to the real traces (set *base*) we generated two more sets of instances,

[2]https://developers.google.com/optimization/

one which is composed by especially computationally intensive jobs, in terms of resource requested (*highLoad*), and one which presents jobs composed by an unusually high number of job-units (*manyUnits*). These additional groups were generated using selected subsets of the original traces. From these sets we randomly selected subsets of smaller instances with dimension of 25, 40, 50, 60, 80, 100, 150 and 200 jobs; for each size we used 50 different instances in our tests. The instances of dimension 25 and 50 were run on the smaller machine A while the remaining instances executed on machine B.

On each instance we ran the PRB algorithm (*PRB*), the hybrid approach with no feedback iteration (*DEC_noFeedBack*) and the hybrid approach with feedback (*DEC_feedBack*). We tested the hybrid approach both with and without the interaction between the two layers because the method without feedback is much faster than the one with the interaction, therefore better suited to a real-time application as a HPC dispatcher. The CP component of the decomposed method has a timeout which forces the termination of the search phase. The timeout is set to 5 seconds; if the solver finds no solution within the time limit, the search is restarted with a increased timeout (we multiply by 2 the previous timeout), until we reach a maximum value of 60 seconds - which is actually never reached in our experiments. The *PRB* is extremely fast as finding a solution takes only a fraction of second even with the larger instances; *DEC_noFeedBack* requires up to 3-4 seconds with the larger instances (but usually a lower quality solution is found in less than a second) and the *DEC_noFeedBack* requires significant larger times to compute a solution due to the multiple iterations, in particular up to 15-20 seconds with the instances of 200 jobs.

Each experiment was repeated with different values of power capping to explore how the bound on the power influences the behaviour of the developed dispatcher.

At every run we measured the average weighted queue time of the solution, that is the average time spent waiting by the jobs, weighted with the expected wait time (given by the queue of the job). As a unit of measure we use the *Expected Wait Time*, (EWT), i.e. the ratio between the real wait time and the expected one. A EWT value of 1 tells us that a job has waited exactly as long as it was expecting; values smaller than 1 indicate that the job started before the expected start time and values larger than one that the job started later than expected. To evaluate the performance of the different approaches we then compute the ratio between the average weight queue time obtained by *PRB* and by the two hybrid methods; finally we plot these ratios in the figures presented in the rest of the section. Since the best average queue times are the lowest ones, it is clear that if the value of the ratio goes below one the *PRB* approach is performing better, while when the the value is above one then the hybrid approaches are obtaining better results.

The following figures will show the ratios in the y-axis while the x-axis will specify the power capping level; we only show significant power capping levels, i.e. the one larger enough to allow a solution to the problem, hence the x-scale range may vary.

Machine with 8 nodes Figures 1, 2 and 3 show the results of the experiments with the machine with 8 nodes; each figure corresponds respectively to the *base* workload set of instances (both size 25 and 50 jobs), the *highLoad* case and finally the *manyUnits* case. The solid lines represent the ratios between the average queue times obtained by *PRB* and those obtained by *DEC_feedBack*; conversely, the dashed line is the ratio between *PRB* and *DEC_noFeedBack*. As we can see in Figure 1 with an average workload the hybrid approaches usually outperform the heuristic algorithm, markedly in the 25 jobs case and especially at tighter power capping. With less tight power capping levels usually the hybrid approaches and *PRB* offer more similar results; this is reasonable since when the power constraints are more relaxed the allocation and scheduling decisions are more straightforward and the more advanced reasoning offered by CP is less necessary.

Figure 1: 8 Node - *Base* Set

It is easy also to see that the hybrid method with the feedback mechanism always outperforms the feedback-less as we expected: the feedback-less solution has always inferior quality w.r.t. to those generated by the method with the interaction - the solution produced by *DEC_feedBack* is the same produced by *DEC_noFeedBack*. Our focus should be on the extent of the improvement guaranteed by the feedback mechanism in relation to the longer time required to reach a solution. For example, Fig.1 (corresponding to the original Eurora workloads) shows that the feedback method offers clear advantages, in particular with tighter power constraints (around 10%-15% gain over the feedback-less one), a fact that could justify its use despite the longer times required to reach a solution. Conversely Figure 2 reveals that the two hybrid approaches offer very similar performance if the workload is very intensive, leading us to prefer the faster *DEC_noFeedBack* in these circumstances in case of the implementation of the real dispatcher.

Figure 2: 8 Node - *HighLoad* Set

If we consider the workload characterized by an unusually high amount of job-units per job we can see slightly different results: as displayed in Figure 3, *DEC_feedBack* definitely outperforms the other two methods with tight power capping values, especially in the case of instances of 25 jobs. When the power constraints get more relaxed the three approaches offers almost the same results.

Figure 3: 8 Node - *ManyUnits* Set

Machine with 32 nodes In Figures 4 and 5 we can see the results of some of the experiments done on the machine with 32 nodes (in particular we present the case size 40 and 100); we present only a subset of the experiments made due to space limitations, but the results not shown comply with those presented here.

Figure 4 shows again the comparison between the two hybrid approaches and the heuristic technique in the case of average workload. The pattern here is slightly different from the 8-nodes case: after an initial phase where the hybrid methods perform better, *PRB* offers better results for intermediate levels of power capping (around 3000W); after that we can see a new gain offered by the hybrid techniques until we reach a power capping level around 6000W (the power constraint relaxes), where the three approaches provide more or less the same results. It is evident again that the method with feedback outperforms the feedback-less one, especially with the larger instances.

Figure 4: 32 Node - *Base* Set

In Figure 5 we can see the results obtained with the computationally more intensive workload. In this case *PRB* performs generally better at lower power capping levels until the power constraint become less tight and the three methods produce similar outcomes again.

Figure 5: 32 Node - *Highload* Set

We have performed a number of additional experiments, not shown here, and we could summarize that with the machine with 32 nodes with average workloads the hybrid approaches perform definitely better than the heuristic technique for most power capping levels except a small range of intermediate values (up to a 60% reduction of average queue times). On the contrary, if we force very intensive workloads we obtain better outcomes with *PRB*, even though with usually smaller, but still significant, differences (around 10%). We

can also see that with these intensive workloads the difference between the two hybrid approaches is minimal and this suggest that the basic feedback mechanism needs to be refined.

7 Conclusions

In this paper we dealt with the allocation and scheduling problem on Eurora HCP system subject to the power consumption constraint - power capping. We presented two approaches to solve this problem: 1) a heuristic algorithm and 2) a novel, hybrid approach which combines a CP model for scheduling and a heuristic component for the allocation. We compared the two approaches using the average queue times as an evaluation metrics. Short waiting times correspond to a higher quality of service for the system users.

We tackled a complex problem due to the limited amount of multiple, heterogeneous, resources and the additional constraint introduced by the finite power budget. In addition to the complexity of the problem (scheduling and allocation are NP-Hard problems) we also had to consider the real-time nature of the application we wanted to develop, thus we had to focus on methods able to quickly produce good solutions. In this preliminary study we shown that the quality of the solution found by the different approaches varies with the levels of power capping considered.

As a long-term goal we plan to further develop the preliminary power capping feature presented here in order to integrate it within the actual dispatcher we already developed and implemented on Euora. In order to do that we will need to develop methods to allow our approach to operate quickly enough to match the frequency of job arrivals; the heuristic would be already fast enough but the hybrid approach will require us to research new techniques in order to cope with the real-sized version of Eurora - and possibly even larger systems.

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

References

[1] N. , Mudge. . In P. Culler, David E. Druschel, editor, *OSDI*. USENIX Association, 2002. Operating Systems Review 36, Special Issue, Winter 2002.

[2] A. Auweter, A. Bode, M. Brehm, L. Brochard, N. Hammer, H. Huber, R. Panda, F. Thomas, and T. Wilde. A case study of energy aware scheduling on supermuc. In J. Kunkel, T. Ludwig, and H. Meuer, editors, *Supercomputing*, volume 8488 of *Lecture Notes in Computer Science*, pages 394–409. Springer International Publishing, 2014.

[3] A. Banerjee, T. Mukherjee, G. Varsamopoulos, and S. K. Gupta. Cooling-aware and thermal-aware workload placement for green hpc data centers. In *Green Computing Conference*, pages 245–256, 2010.

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

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

[6] A. Bartolini, A. Borghesi, T. Bridi, M. Lombardi, and M. Milano. 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*, pages 765–780, 2014.

[7] A. Bartolini, M. Cacciari, C. Cavazzoni, G. Tecchiolli, and L. Benini. 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*, March 2014.

[8] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini. Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller. *IEEE Trans. Parallel Distrib. Syst.*, 24(1):170–183, 2013.

[9] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick. Exascale computing study: Technology challenges in achieving exascale systems, September 2008.

[10] R. Chu, R. Simons, M. Ellsworth, R. Schmidt, and V. Cozzolino. Review of cooling technologies for computer products. *Device and Materials Reliability, IEEE Transactions on*, 4(4):568–585, Dec 2004.

[11] F. Collina. Tecniche di workload dispatching sotto vincoli di potenza. Master's thesis, Alma Mater Studiorum Università di Bologna, 2014.

[12] C. Conficoni, A. Bartolini, A. Tilli, G. Tecchiolli, and L. Benini. Energy-aware cooling for hot-water cooled supercomputers. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 1353–1358, San Jose, CA, USA, 2015. EDA Consortium.

[13] J. J. Dongarra. Visit to the national university for defense technology changsha, china. Technical report, University of Tennessee, June 2013.

[14] J. J. Dongarra, H. W. Meuer, and E. Strohmaier. 29th top500 Supercomputer Sites. Technical report, Top500.org, Nov. 1994.

[15] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[16] W.-c. Feng and K. Cameron. The Green500 List: Encouraging Sustainable Supercomputing. *IEEE Computer*, 40(12), December 2007.

[17] A. Gandhi, M. Harchol-Balter, R. Das, J. O. Kephart, and C. Lefurgy. Power capping via forced idleness. 2009.

[18] R. Haupt. A survey of priority rule-based scheduling. *Operations-Research-Spektrum*, 11(1):3–16, 1989.

[19] J. Kim, M. Ruggiero, and D. Atienza. Free cooling-aware dynamic power management for green datacenters. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 140–146, July 2012.

[20] J. Kim, M. Ruggiero, D. Atienza, and M. Lederberger. Correlation-aware virtual machine allocation for energy-efficient datacenters. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1345–1350, San Jose, CA, USA, 2013. EDA Consortium.

[21] J. M. Kim, S. W. Chung, and S. K. Seo. Looking into heterogeneity: When simple is faster.

[22] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. Simunic Rosing. Managing distributed ups energy for effective power capping in data centers. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pages 488–499. IEEE, 2012.

[23] D. Kudithipudi, Q. Qu, and A. Coskun. Thermal management in many core systems. In S. U. Khan, J. Koodziej, J. Li, and A. Y. Zomaya, editors, *Evolutionary Based Solutions for Green Computing*, volume 432 of *Studies in Computational Intelligence*, pages 161–185. Springer Berlin Heidelberg, 2013.

[24] P. Laborie and J. Rogerie. Reasoning with conditional time-intervals. In *Proc. of FLAIRS*, pages 555–560, 2008.

[25] C. Lefurgy, X. Wang, and M. Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11(2):183–195, 2008.

[26] C. L. Pape, P. Couronné, D. Vergamini, and V. Gosselin. Time-versus-capacity compromises in project scheduling. *In Proc. of the 13th Workshop of the UK Planning Special Interest Group*, pages 1–13, 1994.

[27] S. Reda, R. Cochran, and A. K. Coskun. Adaptive power capping for servers with multithreaded workloads. *IEEE Micro*, 32(5):0064–75, 2012.

[28] A. P. Works. Pbs professional®12.2 administrator's guide. http://resources.altair.com/pbs/documentation/support/PBSProAdminGuide12.2.pdf, 2013.

[29] H. You and H. Zhang. Comprehensive workload analysis and modeling of a petascale supercomputer. In W. Cirne, N. Desai, E. Frachtenberg, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 7698 of *Lecture Notes in Computer Science*, pages 253–271. Springer Berlin Heidelberg, 2013.