# Proactive Workload Dispatching on the EURORA Supercomputer

Andrea Bartolini, Andrea Borghesi, Thomas Bridi, Michele Lombardi, and Michela Milano

DISI, University of Bologna, Italy
{a.bartolini,andrea.borghesi3,michele.lombardi2,michela.milano}@unibo.it
thomas.bridi@gmail.com

**Abstract.** In the era of Cloud Computing, Big Data, and Quantum Physics Simulations, data centers play in the world ICT infrastructure a role as big as (sadly) their power consumption. In many cases, a surprising amount of such consumption is due to idle resources, either introduced to face workload peaks or leftovers of workload fragmentation. In this context, proactive workload dispatching has the chance to improve the utilization of computing resources, thus reducing the idle time *and* improving the ability to handle peaks. In this paper, we devise a CP based approach for proactive workload dispatching on the EURORA supercomputer placed at the CINECA computing center in Bologna. The new system is evaluated on simulated job traces, where it leads to remarkable improvements in terms of both machine utilization and waiting times for queued jobs with respect to the currently used dispatcher, i.e., Portable Batch System (PBS).

## 1 Introduction

Computing centers play a key role in modern ICT architectures: they run our internet services, keep track of our savings, make our research possible. They are also well known to be power hungry: in Italy, data centers make for $\sim$2% of the national energy consumption, for a total of 6.6 TWh (roughly that of the Calabria region, according to data by Fondazione Politecnico di Milano, 2010).

The mainstream solution to reduce such a gigantic consumption is to employ efficient hardware or efficient design. By doing so, it is possible to obtain remarkable reductions of the PUE index (Power Usage Effectiveness), i.e. the ratio between the power consumption of the whole data center and the power consumption of the IT equipment alone. Recently, a joint effort by the CINECA inter-university consortium [1] in Italy and the Eurotech group [2] has led to the design of the EURORA system. Thanks to an innovative liquid based cooling system and carefully chosen hardware components, this new machine has a PUE of just 1.05 and managed to reach the top of the Green 500 ranking in the first half of 2013, effectively becoming for a time the most efficient supercomputer on earth. As a comparison, PUE values of around 3 were still common in 2009.

B. O'Sullivan (Ed.): CP 2014, LNCS 8656, pp. 765–780, 2014.
© Springer International Publishing Switzerland 2014

However, reducing the PUE is just a half of the problem. Data by McKinsey [3] for US data centers reveals that on average only 6-12% of the power is employed for actual computation. The reason for this dramatically low value lies in *how efficiently the existing IT resources are used*. In particular, redundant resources are usually employed to maintain the quality of service under workload peaks. More redundant resources are also needed to compensate for the fragmentation resulting from suboptimal dispatching choices. As a consequence, a typical data center ends up packing a lot of idle muscles. Unfortunately, idle resources still consume energy: for a 1MW center with a 1.5 PUE, a 30% utilization means a 1M€ annual cost and 3,500 tons of $CO_2$. In this context, optimization techniques can enable dramatic improvements in the resource management, leading to lower costs, better response times, and fewer emissions.

In this paper, we tackle the problem of performing workload dispatching over the EURORA supercomputer, operating at the CINECA computing center in Bologna. The machine is employed for High Performance Computing (HPC) applications and has a job submission system currently managed by a PBS Dispatcher (Portable Batch System [9]). The dispatcher relies on a number of heuristic techniques to tentatively maintain a high machine utilization and keep the waiting times as small as possible. The CINECA staff has hints that the current system operation could be improved, but finding a more effective PBS configuration is a cumbersome and error-prone task: hence there is interest in alternative approaches. We propose to tackle workload dispatching via proactive scheduling using Constraint Programming. 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.

Our long-term goal is the development of a state of the art workload dispatching approach to replace the current PBS logic. However, at this stage, our main objective is just is to assess the degree of improvement (in terms of waiting times and reduced idleness) that can be obtained by acting on the dispatching decisions. Since our focus is on investigating the solution quality, we do not enforce tight restrictions on the approach run-time (over-exploiting a bit the fact that HPC jobs tend to have large durations). We evaluated our approach by simulating its behavior on real workload traces from the EURORA machine. We compare the results of our approach with those of the currently operating PBS system, demonstrating that substantial improvements are indeed possible.

## 2 System Description and Motivations for Using CP

This section contains a brief presentation of the architecture of the EURORA supercomputer, a discussion about the current dispatching system, and a review to the motivations behind our choice of CP for building an alternative dispatcher.

*The EURORA Supercomputer:* As described in [5] EURORA has a modular architecture based on nodes (blades). In its the current state, the system counts 64 nodes, each one comprising 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. Each node has 16GB of installed RAM memory. EURORA is interfaced with the outside world through a few dedicated computing nodes, physically positioned outside the rack: in particular, a designated login node connects EURORA to the users and runs the job dispatcher (PBS). One of the main boosting factors for the energy efficiency of the supercomputer is the adoption of a hot liquid cooling technology, i.e. the water inside the system can reach up to 50°C. This strongly reduces the energy required for operating the system, since no power is used for actively cooling down the water, and the waste-heat can be recovered as an energy source for other applications.

*The PBS Dispatcher:* The tool currently used to manage the workload on EURORA system is PBS (Portable Batch System), a proprietary job scheduler by Altair PBS Works with the primary duty of allocating computational tasks, 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 request 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 approximate 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*, and are described in Table 1 - for each of those queues we report the maximum number resources that a job could ask if it desires to belong to that queue, i.e. maximum number of nodes, maximum number of cores and GPUs (second column), maximum execution time, and also the approximate time it might wait before starting its execution.

Cyclically, PBS selects a job for execution by polling the state of one or more nodes, trying to find enough available resources to actually start the job execution. If the attempt is unsuccessful, the job is sent back to its queue and PBS proceeds to consider 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 example, the administrators decided to reserve some nodes to the debug queue and to force jobs in the *longpar* queue to start at night.

*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

**Table 1.** Access requirements and waiting times for the PBS queues in EURORA

| *Queue* | *Max Nodes* | *Max Cores/GPUs* | *Max Time* | *Approx. Wait* |
|---|---|---|---|---|
| debug | 2 | 32/4 | 00:30:00 | seconds |
| parallel | 32 | 512/64 | 06:00:00 | minutes |
| longpar | 16 | 256/32 | 24:00:00 | hours |

of an overall plan. Also the hard-coded mapping constraints, designed as a way to ensure low waiting times for specific job classes (e.g. the *debug* queue), may easily cause resource under-utilization, and long waiting times for the remaining jobs (e.g. those in the *longpar* queue). A proactive dispatching approach should intuitively be able to improve the resource utilization and reduce the waiting times without the need of devising such hard-coded restrictions. The task of obtaining a proactive dispatching plan on EURORA can be naturally framed as a resource allocation and scheduling problem, for which CP as a long track of success stories.

## 3 Design of a CP Approach

We adopt a rolling horizon approach, in which our scheduler is awakened whenever a job 1) enters the system or 2) ends its execution. At each iteration, we build a full schedule and mapping for all the jobs in the input queues, taking into account resource capacity limitations. We consider different performance metrics, which we treat either as objective functions or as soft-constraint. Then we dispatch only those jobs that are scheduled for immediate execution.

The schedule is computed based on the worst-case durations (as provided by the users), but the dispatcher reactivation is triggered by the job *actual* terminations (besides of course by their arrivals). Whenever this occurs, the jobs currently in execution cannot be migrated, but all the waiting ones can be re-scheduled to take advantage of the released resources.

### 3.1 Formal Problem Definition

We can now provide a precise definition of the scheduling problem solved at each activation of the dispatcher. 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 systems consider 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.

It is therefore convenient to specify the resource requirements on a job-unit basis. Formally, let $R$ be a set of indexes corresponding to the resource types (cores, memory, GPUs, MICs), and let the capacity of a node $k$ for resource $r \in R$ be denoted as $cap_{k,r}$. We recall that the system has $m = 64$ nodes, each with 16 cores and 16 GB of RAM memory; 32 nodes have 2 GPUs each (and 0 MICs), and the remaining 32 nodes have 2 MICs each (and 0 GPUs). Finally, let $rq_{i,r}$ be the requirement of a unit of job $i$ for resource $r$. The dispatching problem at time $t$ consists in assigning a start time $s_i \geq t$ to each waiting job $i$ and a node to each of its units. All the resource capacity limits should be respected, taking into account the presence of jobs already in execution. Once the problem is solved, only the jobs having $s_i = t$ are actually dispatched.

Informally speaking, in the big picture, the goal is to increase the resource utilization and reduce the waiting times, but those metrics can be meaningfully evaluated only once the actual job durations become known. Hence we formulate the problem in terms of several objective functions that are intuitively correlated with the metrics we are interested in. After extensive preliminary experimentations, we settled for the following possible problem objectives:

$$\max_{i=0..n-1}(s_i + D_i) \qquad \text{(makespan)} \tag{1}$$

$$\sum_{i=0..n-1} \max\left(0, \frac{s_i - q_i - ewt_i}{ewt_i}\right) \qquad \text{(weighted tardiness)} \tag{2}$$

$$\sum_{i=0..n-1} [[s_i - q_i > ewt_i]] \qquad \text{(num of late jobs)} \tag{3}$$

where $n$ is the number of jobs and the notation $[[-]]$ stands for the reification of the constraint between brackets. The makespan has been chosen because compressing the schedule length tends to increase the resource utilization. For the tardiness and the number of late jobs, we consider a job to be late if it stays queued for a time larger than $ewt_i$. The tardiness is weighted, because we assume that users that are already expecting to wait more (i.e. jobs with higher $ewt_i$) should adjust better to prolonged queue times. Both the tardiness based objectives are chosen to improve the perceived response time, in one case by avoiding (proportionally) long waiting times, in the second by reducing the number of jobs in the queues.

### 3.2 CP Model

*Employed CP Techniques:* We defined for the described scheduling problem a CP model that is based on Conditional Interval Variables (CVI, see [8]). A CVI $\tau$ 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 $x(\tau)$. In particular, if $x(\tau) = 0$ the interval is not present and does not affect the model: for this situation we also use the notation $\tau = \perp$.

CVIs can be subject to a number of constraints, including the classical *cumulative* [4] to model finite capacity resources, and the more specific *alternative* constraint [8]. This last global constraint has the following signature:

$$alternative(\tau_0, [\tau_1, .., \tau_{n_\tau}], m_\tau) \tag{4}$$

The constraint forces all the interval variables $\tau_1, \tau_2, \ldots$ to have the same start and end time as $\tau_0$. Moreover, exactly $m_\tau$ of $\tau_1, \tau_2, \ldots$ will be actually present if $\tau_0$ is present. Formally, the constraint enforces:

$$s(\tau_0) = s(\tau_i),\ e(\tau_0) = e(\tau_i) \quad \forall i = 1..n_\tau \qquad \sum_{i=1}^{n_t} x(\tau_i) = m_\tau\, x(\tau_0) \tag{5}$$

*Modeling Decisions and Constraints:* In our model, we use a CVIs to model the scheduling decisions. In particular, we introduce an interval variable $\tau_i$ with duration $D_i$ for each job waiting in the input queues or already in execution. Then, we fix the start of all $\tau_i$ corresponding to running jobs to their real value (which is known at this point). For the waiting jobs we have $s(\tau_i) \in t..eoh$, where $t$ is the time instant for which the model is built and $eoh$ can be given for example by $t$ plus the sum of the maximum duration of all jobs[1]. All the $\tau_i$ variables are mandatory, i.e. $x(\tau_i) = 1$.

Mapping decisions should be taken at the level of single job-units. The modeling style we adopt for them is best explained by temporarily introducing a simplifying assumption, namely that no two units of the same job can be mapped on a single node. With this assumption, the mapping decisions can be modeled by introducing a second set of *optional* interval variables $\upsilon_{i,k}$ such that $x(\upsilon_{i,k}) = 1$ iff a unit of job $i$ is mapped to node $k$.

However, mapping multiple units of the same job on the same node is possible and can be beneficial. To account for this possibility, we have to introduce for each job $i$ multiple sets of $\upsilon$ variables. Specifically, we add one more index and we maintain the semantic, so that we have variables $\upsilon_{i,j,k}$ such that $x(\upsilon_{i,j,k}) = 1$ iff a unit of job $i$ is mapped to node $k$. The $j$ index is only used to control the number of job units that can be mapped to the same node. Finding a suitable range for the index is a critical step: on the one hand, allowing $j$ to range on $0..rn_i - 1$ (i.e. one set of $\upsilon$ variables for each requested node) is a safe choice. On the other hand, it is impossible to map multiple units of the same job on the same node if doing so would exceed the availability of some resource. Hence, a valid upper bound on the number of $\upsilon$ variable sets for a single job $i$ is given by:

$$p_i = \min\left(rn_i, \min_{r \in R}\left\lfloor \frac{cap_{k,r}}{r_{i,r}} \right\rfloor\right) \tag{6}$$

[1] Note that it is possible to shift all the domains by subtracting the smallest $s_i$ to all values, so that at least one $s(\tau_i)$ has a minimum of 0.

and for each job $i$, the index $j$ can range in $0..p_i-1$. Then we have to specify that exactly $rn_i$ job-units should be mapped, i.e. that exactly such number of $v_{i,j,k}$ intervals should be present. This can be done by using an *alternative* constraint:

$$alternative(\tau_i, [v_{i,j,k}], rn_i) \qquad \forall i = 0..n-1 \tag{7}$$

Additionally, the alternative constraint forces all the job-units to start at the same time instant as $\tau_i$. Now, the resource capacity restrictions can be modeled via a set of cumulative constraints:

$$cumulative([v_{i,j,k}], [D_i^{(p_i)}], [r_{i,r}^{(p_i)}], cap_{i,r}) \qquad \forall k = 0..m-1, \forall r \in R \tag{8}$$

where $m$ is the number of nodes and the notation $D_i^{(p_i)}$ stands for a vector containing $D_0$ repeated $p_0$ times, then $D_1$ repeated $p_1$ times, and so on. As mentioned in Section 2 we disregard all the hard-coded constraints introduced by the PBS administrator and we trust the decision making capabilities of our optimization system with providing waiting times as low as possible.

*Handling the Objective Function:* We consider several variants of our dispatching problem, differing one from each other for the considered objective and for the possible presence of soft constraints. First, we have three "pure" models, obtained by adding on top of the presented formulation one of the problem objectives that we have discussed in Section 3.1:

$$\min \max_{i=0..n-1} e(\tau_i) \qquad \text{(makespan)} \tag{9}$$

$$\min \sum_{i=0..n-1} \max\left(0, \frac{s(\tau_i) - q_i - ewt_i}{ewt_i}\right) \qquad \text{(weighted tardiness)} \tag{10}$$

$$\min \sum_{i=0..n-1} [[s(\tau_i) - q_i - ewt_i > 0]] \qquad \text{(num. of late jobs)} \tag{11}$$

Then we consider three "composite" formulations obtained by choosing as a main cost function one of Equations (9)-(11), and then by posting a constraint on the value of the remaining ones. For example, assuming the makespan is the main objective, we get:

$$\min \max_{i=0..n-1} e(\tau_i) \tag{12}$$

$$\text{s.t.} \sum_{i=0..n-1} \max\left(0, \frac{s(\tau_i) - q_i - ewt_i}{ewt_i}\right) \leq \delta_0\, \theta_0 \tag{13}$$

$$\sum_{i=0..n-1} [[s(\tau_i) - q_i - ewt_i > 0]] \leq \delta_1\, \theta_1 \tag{14}$$

The values $\theta_0$ and $\theta_1$ are obtained by solving the pure models corresponding to the constrained functions. The parameters $\delta_0, \delta_1$ allow to tune the tightness of the constraints. The three new composite formulations are loosely inspired by multi-objective optimization approaches and aim at obtaining good solutions according to one global metric (say, resource utilization), while keeping acceptable levels for the other (say, waiting times).

**Table 2.** An example of problem instance

| $i$ | $rn_i$ | $rq_{i,core}$ | $rq_{i,gpu}$ | $rq_{i,mic}$ | $rq_{i,mem}$ | $D_i$ |
|---|---|---|---|---|---|---|
| 000 | 32 | 4 | 1 | 0 | 1000000 | 14000 |
| 001 | 1 | 14 | 1 | 0 | 400000 | 600 |
| 002 | 2 | 4 | 1 | 0 | 200000 | 14400 |
| 003 | 32 | 16 | 0 | 0 | 400000 | 800 |
| 004 | 32 | 3 | 0 | 2 | 800000 | 400 |

**Table 3.** A feasible solution for the instance from Table 2

| $i$ | $s(\tau_i)$ | $v_{i,0,0}$ | $v_{i,0,1}$ | $v_{i,0,2..31}$ | $v_{i,0,32..63}$ | $v_{i,1,0}$ | $v_{i,1,1}$ | $v_{i,1,2..31}$ | $v_{i,1,32..63}$ |
|---|---|---|---|---|---|---|---|---|---|
| 000 | 0 | $\perp$ | 0 | 0 | $\perp$ | $\perp$ | 0 | $\perp$ | $\perp$ |
| 001 | 0 | 1 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| 002 | 600 | 600 | $\perp$ | $\perp$ | $\perp$ | 600 | $\perp$ | $\perp$ | $\perp$ |
| 003 | 0 | $\perp$ | $\perp$ | $\perp$ | 0 | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| 004 | 800 | $\perp$ | $\perp$ | $\perp$ | 800 | $\perp$ | $\perp$ | $\perp$ | $\perp$ |

*Example of a solution:* Let us suppose we have the set of waiting jobs described in Table 2, then a feasible solution to this instance is described in Table 3. As reported in the table, jobs 000, 001 and 002 can execute only on the nodes equipped with GPUs (i.e. node 0 to 31), job 004 can execute only in nodes with MICs (i.e. node 32 to 63). Tow units of job 000 are allocated on node 1, the other 30 units of job 000 are allocated in nodes from 2 to 31; node 0 is completely free and can run job 001 while job 000 is executing; job 003 can execute on nodes from 32 to 63; after the termination of job 001, job 002 can start its execution with two units on node 0 and after the termination of job 003, job 004 can start in nodes from 32 to 63.

## 4 Added Value of CP

The scheduler 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 this stage we (and the CINECA consortium) are interested in investigating the kind of improvements that could be obtained by changing the dispatcher behavior. On this purpose, we have compared the results we obtained with our dispatcher and the ones achieved by PBS as it is currently configured on EURORA.

We performed the comparison on real PBS execution traces, which contain all the information that is usually available at the job arrival times (i.e. the chosen queue, the resource requirements, the maximum execution time). Additionally, the traces report for each job two important pieces of information, namely the *actual* duration (which we use together with the arrival time to simulate the scheduler activation events) and the start time assigned by PBS.

Our approach was implemented using IBM ILOG CP Optimizer [6] using its default search strategy, which is based on Self-adapting Large Neighborhood

**Table 4.** Models comparison, queue times

| | *Average Queue Time* | | | |
|---|---|---|---|---|
| *Model* | *all* | *debug* | *parallel* | *longpar* |
| MKS | 187.14 | 4.77 | 161.81 | 0.01 |
| MKS WT/NL | 165.98 | 0.10 | 160.04 | 0.01 |
| NL | 722.04 | 2.30 | 316.92 | 369.14 |
| NL MKS/WT | 201.32 | 0.31 | 145.59 | 18.99 |
| WT | 662.18 | 2.16 | 203.50 | 446.34 |
| WT MKS/NL | 861.81 | 0.76 | 278.60 | 572.29 |
| PBS | 6840.81 | 17.34 | 2825.05 | 3600.40 |

**Table 5.** Models comparison, system load

| | *Average Resource Utilization* | | | | |
|---|---|---|---|---|---|
| *Model* | *cores* | *GPUs* | *MICs* | *cores (%)* | *Avg jobs* |
| MKS | 678.81 | 45.21 | 3.99 | 66% | 121.68 |
| MKS WT/NL | 701.92 | 45.61 | 3.99 | 68% | 121.92 |
| NL | 614.75 | 45.89 | 3.99 | 60% | 116.58 |
| NL MKS/WT | 670.75 | 45.00 | 3.99 | 65% | 121.21 |
| WT | 671.41 | 47.67 | 3.98 | 66% | 120.50 |
| WT MKS/NL | 620.45 | 41.72 | 3.99 | 61% | 119.07 |
| PBS | 447.98 | 29.16 | 0.33 | 46% | 63.04 |

Search [7] guided by an Linear Programming relaxation. At each scheduler activation we use the best solution found within a time limit to decide the jobs that should start. To allow a fair comparison, all traces were pre-processed to reset the waiting time of all jobs that are in queue at the beginning of the trace, so that this is not taken into account. Additionally, we have subtracted from the PBS waiting times the overhead required for implementing the dispatching decision. This was experimentally identified by analyzing the traces themselves.

### 4.1 Evaluation of Our Models

We performed an evaluation of all our models on a PBS execution trace containing data for a batch of jobs that was considered for dispatching in a 2-hour long interval. The main performance metrics considered are (1) the time spent by the jobs in the queues while waiting their execution to begin (ideally as low as possible), and (2) the overall utilization of the system (ideally as large as possible). Waiting times are measure of the perceived quality of services, while a high utilization directly translates to a low number of idle (but still power consuming) resources.

The results for the first batch (BATCH1) are presented in Table 4 and Table 5; the models evaluated are the three "pure" ones (Makespan [MKS], Weighted Tardiness [WT] and Num. of late jobs [NL]) plus the three composite ones (i.e. with Makespan as main objective and constraints on Weighted tardiness and

**Table 6.** Job traces composition

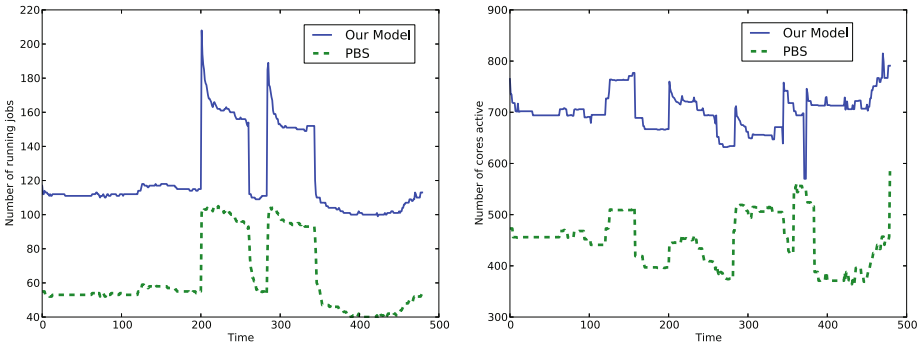
| | *BATCH1* | *BATCH2* | *BATCH3* |
|---|---|---|---|
| #jobs | 437 | 434 | 619 |
| #jobs DEBUG | 237 | 133 | 127 |
| #jobs PAR | 130 | 240 | 415 |
| #jobs LONGPAR | 62 | 25 | 12 |
| #jobs req. GPUs | 85 | 203 | 224 |
| #jobs req. MICs | 3 | 1 | 1 |
| #jobs req. 1 core | 298 | 197 | 258 |
| #jobs req. 2 cores | 2 | 73 | 38 |
| #jobs req. 4 cores | 1 | 4 | 7 |
| #jobs req. 5 cores | 1 | 1 | 0 |
| #jobs req. 6 cores | 6 | 2 | 3 |
| #jobs req. 8 cores | 59 | 56 | 187 |
| #jobs req. 8+ cores | 70 | 101 | 126 |

Num. of late jobs [MKS WT/NL], and similarly for the others). In the table we can see the average waiting time per job (both total and per-queue). There is a remarkable improvement w.r.t PBS for all the models, and those using the Makespan as main objective (MKS and MKS WT/NL). All the composite models perform better than their pure counterparts when dealing with the jobs from *debug* queue (short and with relatively low requirements). The models with Makespan as primary objective do their best when dealing with the long jobs from the *longpar* queue.

The corresponding resource utilization statistics are reported in Table 5, showing for each model and PBS the average number of used cores, GPUs and MICs over time. Again, we can see a significant improvement in comparison to PBS performance, but in this case the differences between our models are less clear. In particular, the average numbers of used GPUs and MICs is very similar – probably because not every job needs an accelerator –, but we can notice that MKS WT/NL is the model which performs a bit better in terms of the average number of active cores. In the fifth column of the table we see the average number of jobs that are in execution at each time instant: more running jobs usually correspond to a higher utilization and a smaller time to complete the execution of the batch. Finally in the last column we report the average percentage of active cores on EURORA, which is a good index for the utilization of the whole system. As one can see, our best results (coming from the MKS WT/NL) are around 20% better than those of PBS. No approach was able to reach a 100% utilization: to a large extent, this appears to be due to the presence of bottleneck resources (e.g. GPUs) and to their allocation.

### 4.2 Comparison with PBS

The previous results show that our best model is a composite one, namely MKS WT/NL, thus such mode was chosen for a more detailed comparison with PBS




(a) Running Jobs (b) Active cores

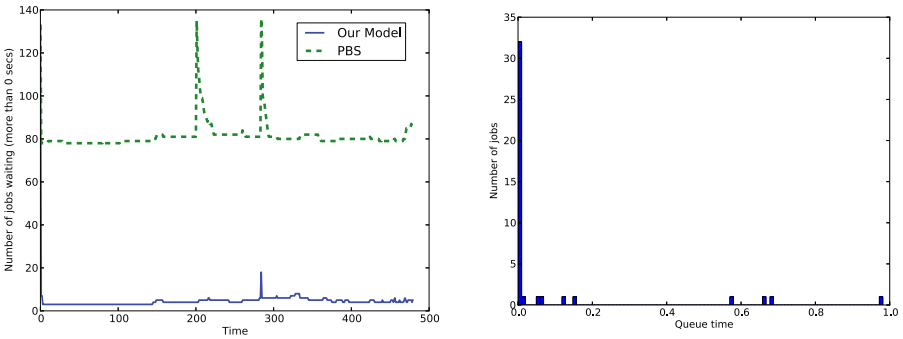
**Fig. 1.** EURORA utilization on the first trace (BATCH1)

on three PBS execution traces, each one corresponding once again to a two-hour time frame of the EURORA activity. The features of the job batches considered in each trace (i.e. BATCH1, BATCH2, BATCH3) are summarized in Table 6, which reports the total number of jobs, the number of jobs in each queue[2], the number of jobs requiring at least one GPU or MIC and the number of jobs requiring a certain number of cores.

We start by presenting the results for BATCH1, which is the same we used for evaluating the model. The jobs considered in this trace belong to a wide range of classes, with different resource requirements and different execution times. In Fig. 1a we can observe the number of active jobs in the considered time frame, for both our approach (solid line) and PBS (dashed line). Fig. 1b reports instead the number of active cores. Our approach significantly outperforms PBS, being able to execute more jobs concurrently and to use a larger fraction of the available cores. Neither approach managed to reach the optimal system usage: this could be due to (a combination of) the presence of bottleneck resources, to suboptimal allocation choices, or simply to the lack of more workload to be dispatched. Fig. 2a shows the number of waiting jobs at each time step for our approach and PBS. From the data in the figure, we can deduce that our approach managed to dispatch most of the incoming jobs immediately, suggesting that the machine underutilization is at least in part to blame on the lack of more jobs. Still, suboptimal choices and resource bottlenecks cause some jobs to wait (a relatively high number of them, in the case of PBS).

Fig. 2b contains a histogram with the waiting times for our model, weighted by the (inverse of) the Estimated Waiting Time of the queue they belong to. The histogram shows how many jobs ($y$-axis) wait for a certain amount of times their $ewt_i$ ($y$-axis). The majority of the waiting jobs with our approach stay in their queue for a very short time, unlike in the case of PBS, where especially the jobs

[2] The sum of those values may be lower than the total, because we do not report detailed statistics for some minor queues.



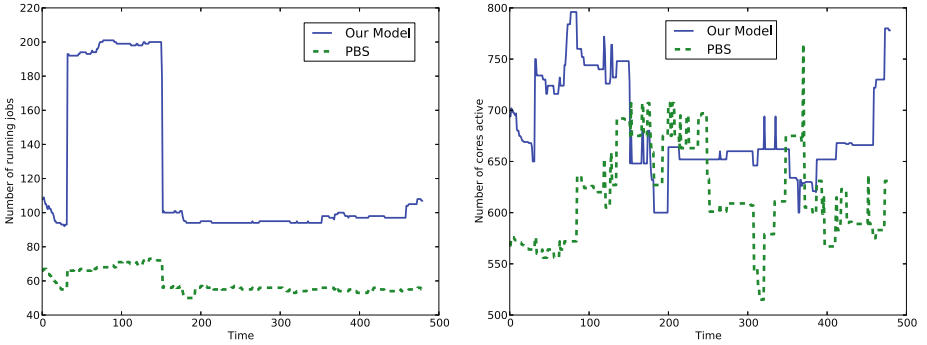

(a) Jobs in Queue (b) Times in Queue

**Fig. 2.** Waiting jobs and queue time for BATCH1

in the *longpar* queue tend to be considerably delayed. We recall that currently these jobs (which are characterized by longer durations than the remaining ones) are forced to execute only at night, for fear or delaying jobs in the *debug* or *parallel* queue. The evidence we provide here leads us to believe that such a strong constraint is in fact not needed when using a proactive approach, and its removal could provide benefits in terms of both queue time and average utilization of the supercomputer resources.

Fig. 3 and Fig. 4 refer instead to our second trace, i.e. to the jobs in BATCH2. This is another mixed group of jobs in terms of computational and resource requirements, but in this case we have many more GPU requests, putting a great strain on the dispatcher since GPUs in EURORA are a much fewer than cores. The consequences of this situation can be observed in Fig. 3a and Fig. 3a, respectively showing the number of running jobs and active cores over time. For both PBS and our model we notice that the number of jobs in execution, after an initial spike, reaches a cap in the middle section of the trace, although the percentage of actives cores is not even close to 100%. This cap occurs because in many cases, basically all waiting jobs are requiring a GPU and hence, even if there are have available cores, they cannot be used. Despite that, we still manage to achieve a largely improved schedule than the one of PBS in term of number of running jobs. In particular, the average number of active GPUs with our dispatcher is higher than 63: given that the whole supercomputer counts only 64 GPUs, this means that the performance obtained by our approach for the GPU-requiring jobs is very close to the theoretical limit.

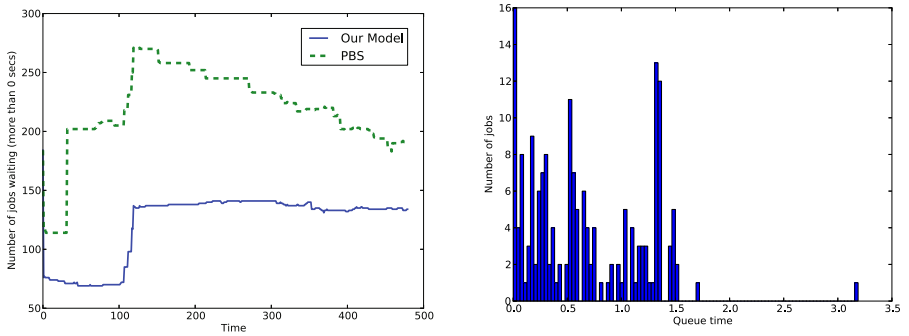
We owe this result to the proactive nature of our scheduler, which allow us to more efficiently use constrained resources. For example, suppose we have node $A$ and $B$, where $A$ has $n$ cores and 1 GPU while $B$ has only $n$ cores, and suppose that A and B are fully occupied by a previous job. We also have *job1* and *job2* waiting to start their execution: *job1* needs $n$ cores and a GPU, whereas *job2* requires only the cores and has higher priority (for PBS). When the job currently




(a) Running Jobs (b) Active cores

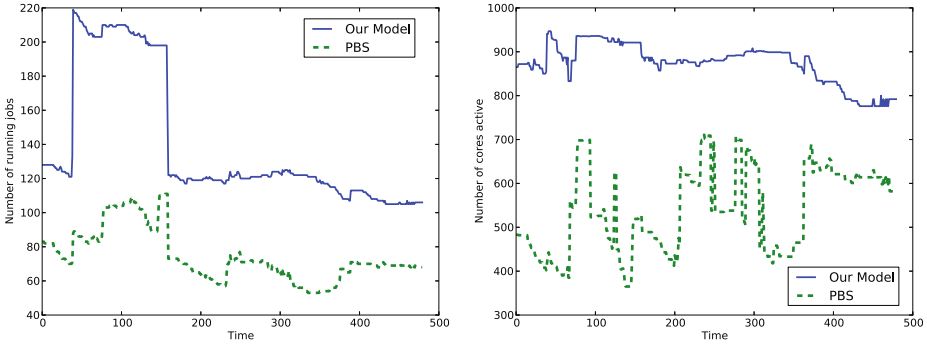
**Fig. 3.** EURORA utilization on the second trace (BATCH2)

occupying nodes $A$ and $B$ terminates, PBS selects *job2*, then it checks if on $A$ there are enough cores to satisfy the requirements. Since this is true in our example, PBS dispatches *job2* on node $A$, using up all node cores and leaving the GPU idle. In this scenario, *job2* cannot start executing until the other job has terminated. Conversely our dispatcher would have made a smarter - and in this particular case obvious - decision, that is putting *job1* on $B$, since it only needs cores, and *job2* on $A$, without further delay. In Figure 4 we can see our performance in terms of queue times for BATCH2. We outperform PBS again but at the same time we notice how the number of jobs in queue (Fig 4a) follow a similar pattern in both systems, with a distinctive spike after a relatively low initial value: this happens because of the congestion on the GPUs resources we mentioned earlier – after all, optimization can provide improvement only as long as spare resources are available.



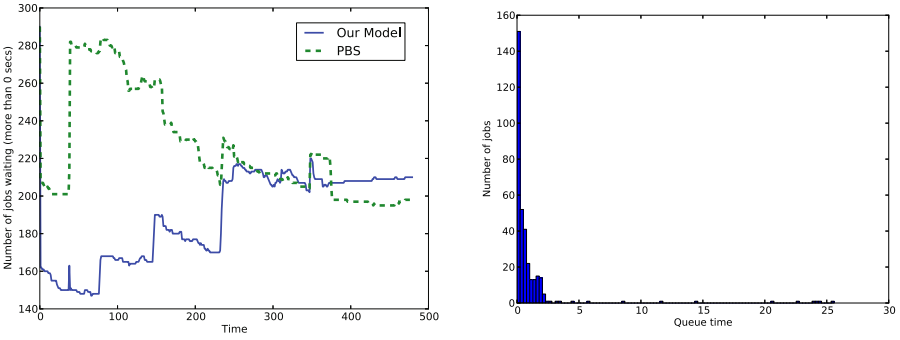

(a) Jobs in Queue (b) Times in Queue

**Fig. 4.** Waiting jobs and queue time for BATCH2




(a) Running Jobs (b) Active cores

**Fig. 5.** EURORA utilization on the third trace (BATCH3)




(a) Jobs in Queue (b) Times in Queue

**Fig. 6.** Waiting jobs and queue time for BATCH3

Finally, we can eventually consider BATCH3 and the results are displayed in Fig. 5 and Fig. 6. The jobs considered in this trace require, on average, a higher number of cores than all other traces and, for a large part, were submitted to the *parallel* queue. They require proportionally fewer GPUs than the jobs in BATCH2, but still more than BATCH1. We manage again to obtain a better usage of computational resources on EURORA, as revealed in Fig. 5b and from the average percentage of actives cores (85% in our model versus 55% with PBS). One more time, these results are due to a smarter management of the different types of resources, although the limitations imposed by the relatively low number of available GPUs still has an impact on the number of running jobs (Fig. 5a). In Figure 6a we can see our model is able not to force to wait as many jobs as PBS, but only during the first half of the trace, while after that point the number of jobs in queue is comparable between the two dispatchers. One possible

explanation for this is again the limit imposed by the GPUs availability, given that not all the cores are occupied, which forces more jobs to wait when a certain threshold for the number of GPUs required is reached.

## 5 Conclusions

In this paper have presented a CP based proactive workload dispatcher for the HPC EURORA supercomputer and compared its performance with those of the system currently in use (PBS). Our goal is to manage the computational resources on the platform so as to achieve a twofold result: increase the machine utilization and then reduce the job waiting times. A higher machine utilization translates into a lower consumption from idle resources and a large number of accepted jobs, with benefits for the supercomputer owner and on the environmental side. Short waiting times correspond to a higher quality of service for the system users.

The problem we tackled was not an easy one, owing to the need to manage multiple objectives and to the limited availability of multiple, heterogeneous, resources. In both the considered metrics (machine utilization and waiting times) we considerably outperformed the current scheduler, showing that there are great margins for improvement when a proactive approach is used. The current, fundamentally reactive approach currently in use proved to have particular difficulties with the simultaneous management of different classes of resources (e.g. cores and GPUs). As a future long-term goal, we plan to further develop our model to replace (or at least complement) the scheduler currently in use on EURORA, with focus on improving its energetic behavior. To achieve this result, we will need to research and develop techniques to allow our approach to operate quickly enough to match the frequency of job arrivals. Moreover, we will need to make some adjustments to take into account the complex policies which regulate exactly the services provided by the supercomputer to its users.

**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. Cineca inter-university consortium web site, `http://www.cineca.it//en` (accessed: April 14, 2014)
2. Eurotech group web site, `http://www.eurotech.com/en/` (accessed: April 14, 2014)
3. Ny times article about a survey by mc kinsey & co., `http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html` (accessed: April 14, 2014)
4. Baptiste, P., Laborie, P., Le Pape, C., Nuijten, W.: Constraint-based scheduling and planning. Foundations of Artificial Intelligence 2, 761–799 (2006)

5. Bartolini, A., Cacciari, M., Cavazzoni, C., Tecchiolli, G., Benini, L.: Unveiling eurora - thermal and power characterization of the most energy-efficient supercomputer in the world. In: Design, Automation Test in Europe Conference Exhibition (DATE) (March 2014)
6. Laborie, P.: IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 148–162. Springer, Heidelberg (2009)
7. Laborie, P., Godard, D.: Self-adapting large neighborhood search: Application to single-mode scheduling problems. In: Proc. of MISTA (2007)
8. Laborie, P., Rogerie, J.: Reasoning with conditional time-intervals. In: Proc. of FLAIRS, pp. 555–560 (2008)
9. Altair PBS Works. Pbs professional®12.2 administrator's guide (2013), http://resources.altair.com/pbs/documentation/support/PBSProAdminGuide12.2.pdf