

# A CP Scheduler for High-Performance Computers

Thomas Bridi, Michele Lombardi, Andrea Bartolini, Luca Benini, and Michela Milano

{thomas.bridi,michele.lombardi2,a.bartolini,luca.benini,michela.milano}@unibo.it

Università di Bologna

**Abstract.** Scheduling and dispatching tools for High-Performance Computing (HPC) machines have the role of mapping incoming jobs to the available resources, trying to maximize equipment utilization and user satisfaction. Optimal Job Scheduling is a well-known NP-hard problem, forcing commercial schedulers to adopt greedy approaches based on rules. Constraint Programming (CP) is a well-known combinatorial optimization approach that has been shown to be very effective in optimally solving scheduling problems. We present the first CP-based job scheduler for HPC machines, working in a real-life production environment. We evaluate our solution both on a cluster of virtual machines and on the Eurora Supercomputer with production workloads. Results show significant improvements in terms of user fair-waiting without degradation in overall machine utilization w.r.t state-of-the-art rule-based dispatchers.

**Keywords:** Constraint Programming, Scheduling, Supercomputer

## 1 Introduction

Today high-performance computing (HPC) centers are investment-intensive facilities with short depreciation cycles. An average supercomputer reaches full depreciation in just three years, hence its utilization has to be aggressively managed to produce an acceptable return on investment. A key role in this challenge is played by scheduling software that decides where and when a job submitted by a user has to be started. The scheduling software orders the set of jobs and the set of nodes and then tries to allocate each job to nodes taking into account the amount of currently available computing resources. When a new job is submitted, the software usually applies a back-filling algorithm that tries to place the job on unused resources without delaying the start of highest priority jobs in queues. These priority-rule-based algorithms are simple and reasonably fast, but they usually do not find the best solution of the scheduling problem. One of the most widespread queue-based scheduling software in HPC facilities is PBS Professional [5]. In this work, we present a complete CP model for solving the optimal scheduling problem in HPC machines. The model significantly extends the one in [1] to account for multiple classes of jobs and their temporal constraints.

In addition, the solution space exploration strategies have been optimized for on-line use, taking into account the impact of the schedule computation time on machine utilization. The CP solver based on the new model has been embedded as a plug-in module within the software framework of a well-known commercial HPC scheduler [5] replacing its scheduling engine. By linking our solver with a state-of-the-art HPC scheduling tool, we have been able to validate our approach on a real-life HPC machine, Eurora from CINECA (Consorzio Interuniversitario Calcolo Automatico). Experiments on Eurora demonstrate that the new scheduler achieves significant improvements in job waiting time with respect to the commercial scheduler used in CINECA, while at the same time maintaining high machine utilization. An experimental campaign on a wide range of synthetic workloads proves that the approach is flexible, robust and well-suited for integration in a portfolio of scheduling strategies to cover different levels of average machine utilizations. In section 3 we show an overview of the scheduling software running on the Eurora HPC machine. In section 4 we formally describe the problem of scheduling. In section 5 we describe the optimization techniques used to model the problem. In section 6 we present our optimization model and all the features implemented to make it desirable for a real HPC center. In section 7 we show results from simulations and from the Eurora supercomputer and we report statistics on the computational overhead. Finally in section 8 we show our conclusions.

## 2 Related work

The problem of batch scheduling is well-known and widely investigated. The interested reader can refer to [4] for a good survey of the scheduling algorithms used in HPC and computing clusters. Most of the algorithms described in [4] can be implemented within commercial scheduling software by defining appropriate “scheduling rules”. To the best of our knowledge, the only examples that apply optimization techniques to a scheduler in a production context is [2]. In this paper, the author present an optimization technique applied as an extension to the TORQUE scheduler. This extension replaces the scheduling core of the framework with a backfilling-like algorithm that inserts one job at a time into the schedule starting from a previous solution and then applies a Tabu Search to optimize the solution. This approach considers a job as a set of resources. This assumption drastically decreases the flexibility of the scheduler by avoiding the possibility for a job to request more than one node. In our work, instead, we consider a job as a set of nodes, each requiring a set of resources. In this way we maintain the flexibility of commercial schedulers (like TORQUE and PBS Professional) but we deal with a more complex settings w.r.t. [2].

## 3 Eurora, Heterogeneous Supercomputer

Eurora is a heterogeneous HPC machine of CINECA. Eurora is composed of 65 nodes, one login node with 12 cores, 32 nodes with 16 cores at 3.1GHz and

2 GPU Kepler K20 each and 32 nodes with 16 cores at 2.1GHz and 2 Intel Xeon phi (MIC) each. Users from this HPC machine can submit a job that specifies the amount of resources, nodes and walltime to a queue; a queue is the place where job waits to be executed. Each queue has a name and a priority, after the submission. The scheduling software decides the start time and nodes where to execute the job. The scheduling and dispatching software currently used in Eurora is PBS Professional 12 from Altair; PBS Professional is a Portable Batch System [5] that schedules jobs based on rules. The original scheduler *PBS\_sched* can be disabled and replaced by with ad-hoc scheduling algorithms. We take advantage of this functionality to implement in a plug-and-play fashion our optimized scheduling policy.

## 4 The Scheduling problem

In this section, we formally describe the problem of on-line scheduling and dispatching of a supercomputer. The scheduling problem considers a set of jobs  $J$  and a set of queues  $Q$ . Every  $job_i$ , defined on the set  $J$ , is submitted in a specific queue  $q_i$  defined on the set  $Q$ . Each job, when submitted, has to specify its maximal duration  $d_i$ , the number of jobs units  $u_i$  (the number of virtual nodes required for the execution) and the amount of required resource  $r_{ijkl}$  (cores, memory, GPUs, MICs) for each job unit  $k \in [1..u_i]$  and for each  $l \in R$ , where  $R$  is the set of resource. Each node  $n_j$  of the system has a limit  $rl_{jl}$  for each resource  $l$ , with  $j \in N$  where  $N$  is the set of nodes. We have to assign the starting execution time  $s_i$  and for each job unit  $ju_{ik}$  of the job  $job_i$ , the node  $n_j$  where it has to be executed. Given the current time  $t$ , the set of running jobs cannot be changed (migration is not supported), while the set of waiting jobs has to be allocated and scheduled on resources without exceeding their capacity at any point in time.

## 5 Constraint Programming

The technique used in this work to model the problem is Constraint Programming. A Constraint Program is defined on a set of variables, each defined on a discrete domain, and a set of constraints. Differently from Convex Optimization (like LP, ILP, etc...), with this paradigm we are not forced to have a convex polytope as solution set and a convex objective function. The global constraint we will use are:

- *alternative*( $a, [b], C$ ) : the constraint holds iff at least  $C$  activities from the vector  $[b]$  has the same start time and duration of the activity  $a$ .
- *cumulative*( $[s], [d], [r], L$ ) : the constraint holds iff all the activities  $i$  defined by a starting at time  $s_i$ , a duration  $s_i$  and a resource requirement  $r_i$  never exceed the resource capacity  $L$  at any point in time.
- *noOverlap*( $a, t, [s], [d]$ ) : the constraint holds iff all the activities  $i$  with start time  $s_i$  and duration  $d_i$  do not overlap an activity with start time  $a$  and duration  $t$  (for each  $i$  we can have  $a \geq s_i + d_i$  OR  $a + t \leq s_i$ ).

- *synchronize*( $a, [b]$ ) : the constraint holds iff the start time  $a$  is synchronized with each start time  $i$  of the vector  $[b]$ .

## 6 CP Model

Starting from the work in [1] we create a model that contains all the requirements and services needed by supercomputers in production. For every job  $i$  we have a Conditional Interval Variables (CVI, see [3])  $job_i$ . A CVI represents an interval variable. The domain of a CVI is a subset of  $\{\perp\} \cup \{[s, e] | s, e \in Z, s \leq e\}$ . If a CVI has domain  $\perp$ , this variable does not belong to the model and it is not considered in the solution process. A job's CVI contains the job walltime  $d_i$  specified by the user. For every job we have also a matrix  $UN_i$  of  $M \times P_{ij}$  of CVIs, where  $M$  is the number of nodes in the system and  $P_{ij}$  is the maximum number of job units dispatchable in the  $j$ th node. These elements assume the value  $s(i)$  if the  $i$ th job uses the node  $j$ , the value bottom otherwise.  $R$  is the set of resources of the node,  $A$  the set of jobs in a queue and  $B$  the set of running jobs. The base model created is described in 1. With the alternative constraint, we introduce the possibility for every job unit to be displaced in a node partially used by another job unit of the same job. The cumulative constrain the set of jobs start times.

$$\begin{aligned}
 & job_i \geq t \quad \forall i \in A \\
 & job_i = s(b) \quad \forall i \in B \\
 & alternative(job_i, UN_{ijk}, u_i) \quad \forall i = 1..N \\
 & cumulative(UN_{ijk}, d_i^{P_{ij}}, r_{ijkl}^{P_{ij}}, rl_{jl}) \quad \forall k = 1..M, l \in R
 \end{aligned} \tag{1}$$

Equation 2 represents the objective function used in this model. This function takes the job waiting-time weighted on the expected waiting time ( $ewt_i$ ) of the queue where the job is submitted. This objective function is designed to optimize the jobs waitings paying attention to the fairness of these. This mean that waitings have to be distributed taking into account the priority of the jobs.

$$\min z = \sum_{i=1}^n \frac{s_i - q_i}{ewt_i} \tag{2}$$

## 7 Experimental Results

We have evaluated the performance of our scheduler in two distinct experimental setups, namely (1) in a simulated environment on Virtual Machines (VM); and (2) on the actual Eurora HPC machine. The PBS software can be configured in different modes to suit the purpose of the system administrator. the following experiments consider two different PBS setups:

1. The CINECA PBS configuration (referred to as PBSFifo): this setup uses a FIFO job ordering, no preemption, and backfilling limited to the first 10 jobs in the queue.

	Test 1			Test 2		
	PBSFifo	PBSWalltime	CP Sched.	PBSFifo	PBSWalltime	CP Sched.
WQT	152,94	137,74	119,77	1034,2	853,681	2441,3
NL	65	60	46	234	200	376
TR	1298810	1223690	1003970	16798300	13693000	16774800
AO	0,47	3,14	11,45	1,02	15,47	34,82

Table 1: Test 1 and Test 2 results

2. A PBS configuration (referred to as PBSWalltime) designed to get the best trade-off between waiting time and computational overhead: this setup employs a strict job ordering (by increasing walltime), no preemption and back-filling limited to the first 400 jobs.

### 7.1 Simulation-based tests

We have designed the simulation so as to evaluate the performance of our CP scheduler w.r.t. PBS. The experiments differ under a wide range of conditions with respect to number of jobs, job units, and platform nodes. The goal is to assess the scalability of both approaches and their ability to deal with workloads having different resource requirements and processing times. The quality of the schedules was measured according to a number of metrics. Specifically, we have defined:

- *Weighted queue time (WQT)*: sum of job waiting-times, each divided (for fairness) by the maximum wait-time of the job queue.
- *Number of late jobs (NL)*: the number of jobs exceeding the maximum wait-time of their queue.
- *Tardiness (TR)*: sum of job delays, where the delay of a job is the amount of time by which the maximum wait-time of its queue is exceeded.
- *Average overhead (AO)*: average computation time of the scheduler.

In test 1 we simulate all the 65 Eurora nodes: the results are in Table 1. Our model manages to outperform considerably PBSFifo and PBSWalltime in terms of all the metrics related to waiting time and delay. In test 2 tested a 65 nodes configuration with a larger number of jobs (namely 700): the results are reported in Table 1. Due to the large number of jobs and (more importantly) job units, in this case, our framework was forced to employ the overhead reduction techniques. Such techniques are indeed effective in limiting the overhead, but they also have an adverse effect on the quality of the model solutions. As it can be seen in the table, our model yields a small improvement in tardiness w.r.t. PBSFifo, a small increase in the total time in queue, and a considerable increase of the number of late jobs, the WQT, and the weighted tardiness.

### 7.2 Execution on Eurora

Thanks to our modeling and design from Section 6, we have managed to obtain a scheduling system that is mature enough to be deployed and evaluated on the

actual Eurora HPC machine. In detail, we have compared the performance of our approach and the PBSFifo configuration over five weeks of regular utilization of the HPC machine. Since the comparison is performed in a production environment, it is impossible to guarantee that the two approaches process the same sequence of jobs. For this reason, we chose to compare the CP approach and PBSFifo in terms of: (1) the average WQT per job, and (2) the average number of used cores over time (i.e. the average core utilization). Our CP system performed consistently better with an average WQT per job of  $\sim 2.50 \cdot 10^{-6}$ , against the  $\sim 3.93 \cdot 10^{-6}$  of PBSFifo. The standard deviation for the two approaches is very similar. The average core utilization obtained by both approaches during each week, show that the two approach have similar performance, which ranges between 520 and 599 for PBSFifo and between 510 and 573 for CP.

## 8 Conclusion

In this paper we presented a scheduler, based on Constraint Programming techniques, that can improve the results obtained from commercial schedulers highly tuned for a production environment and we implemented all the features for made it usable on a real-life HPC setting. The scheduler has been tested both in a simulated environment and on a real HPC machine with promising results. We have seen that in the medium hardness range we can improve results obtained by the commercial scheduler by a significant amount (21% on 152 points of WQT) and in the high hardness range we did not get an improvement due to the computational time and the too aggressive technique we had to implement. The experimental results on the Eurora HPC system shown an improvement on the weighted queue time while the system utilization. Future work will focus on the following directions: improving the integration between the scheduling management framework and the optimizer, and developing incremental strategies to hot-start the optimization engine.

## References

1. Bartolini, A., Borghesi, A., Bridi, T., Lombardi, M., Milano, M.: Proactive workload dispatching on the eurora supercomputer. In: OSullivan, B. (ed.) *Principles and Practice of Constraint Programming, Lecture Notes in Computer Science*, vol. 8656, pp. 765–780. Springer International Publishing (2014)
2. Chlumsky, V., Klusáček, D., Ruda, M.: The extension of torque scheduler allowing the use of planning and optimization in grids. *Computer Science* 13, 5–19 (2012)
3. Laborie, P., Rogerie, J.: Reasoning with conditional time-intervals. In: *Proc. of FLAIRS*. pp. 555–560 (2008)
4. Salot, P.: A survey of various scheduling algorithm in cloud computing environment. *International Journal of research and engineering Technology (IJRET)*, ISSN pp. 2319–1163 (2013)
5. Works, P.: *Pbs professional 12.2, administrators guide*, november 2013