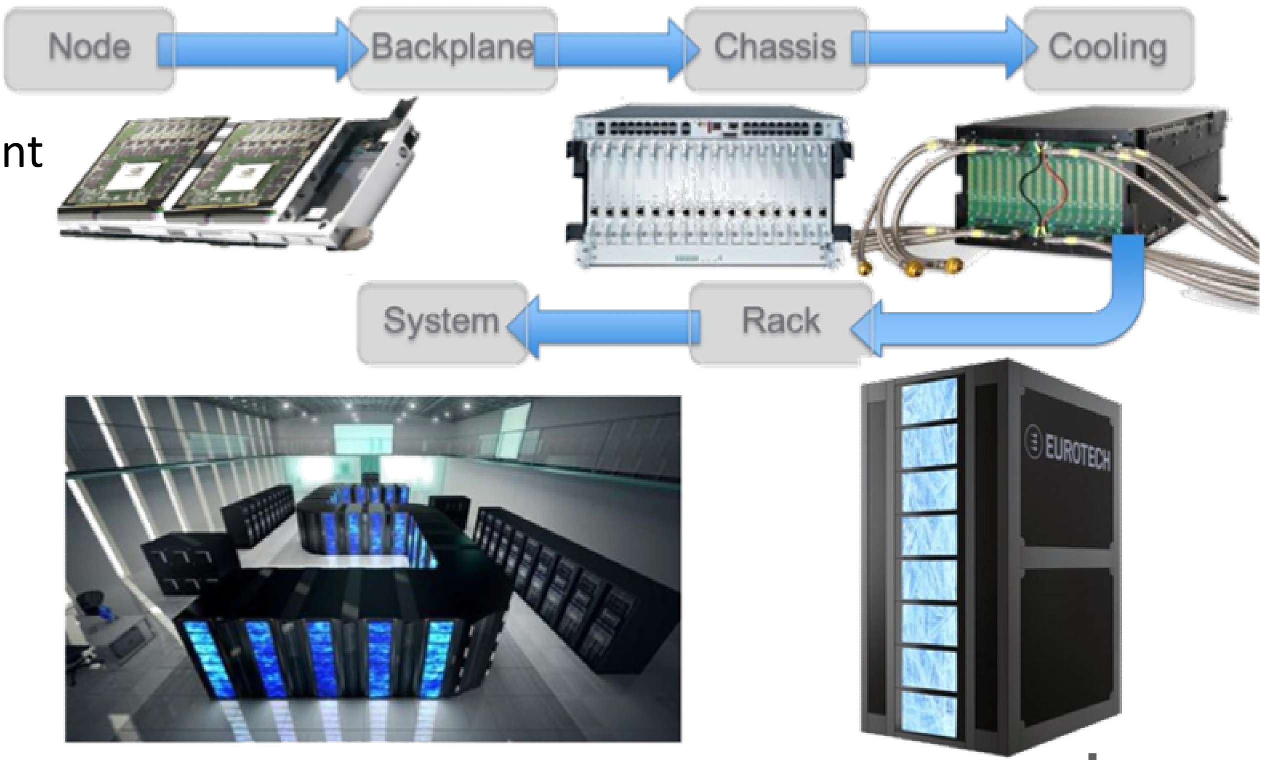
A CP Scheduler for High-Performance Computers

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

Context

- Todays HPC machines have a cost that varies between 3M $ (Eurora HPC) and 390M $ (Tianhe-2 HPC)
- An average supercomputer reaches full depreciation in three to five years
- The challenge is to produce an acceptable return of investment
- A key role in this challenge is played by scheduling software
- Even a relative improvement in utilization, throughput, and quality of service translates in significant return of investments



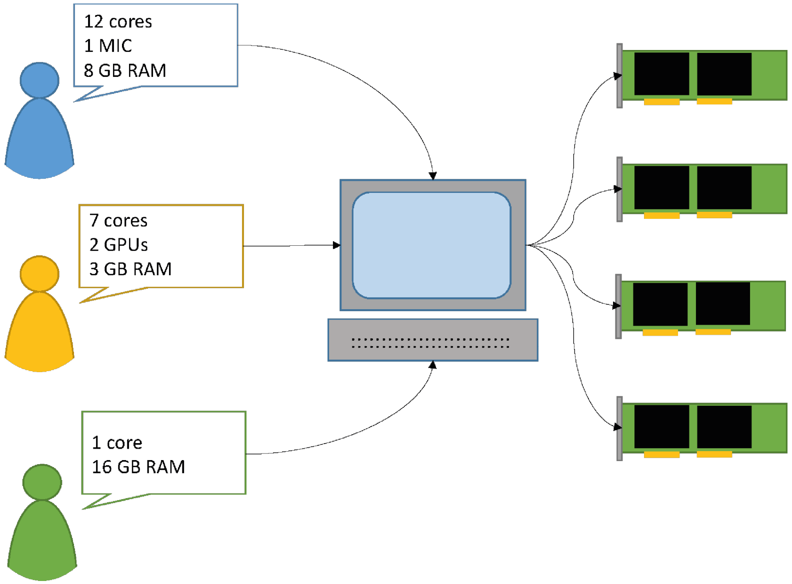
Eurora HPC (CINECA)

- Prototype for future Tier-0 HPC
- TOP Green 500 HPC in June 2013
- Heterogeneous:
  - 32 nodes with 2x 8-cores 3.2GHz Intel E5, 2x Nvidia Kepler K20 (GPU), and 16GB RAM
  - 32 nodes with 2x 8-cores 2.1GHz Intel E5, 2x Intel Xeon Phi (MIC), and 16GB RAM
  - 1 login node 2x 6-cores 2.1GHz Intel E5, and 128GB RAM
- In use scheduler: PBS Professional 12.2

Scheduling Problem

- Set of jobs, each job is composed by job units, each job unit can require:
  - CPUs
  - MICs
  - Memory
  - Nodes with a certain cpu frequency
  - Nodes with a certain hostname

  Every job unit of the same job have same wall-time and start-time
- Set of nodes with physical and virtual resources
- The scheduler have to assign a start-time for each job and a node for each job unit in order to never overutilize resources while keeping high utilization, throughput, and low waitings



Constraint Programming

- Constraint Programming is a programming paradigm to solve Optimization problem
- Easily model scheduling problems
- Usually used for off-line scenario due to the high computing time to solve NP-Hard problems
- Due to the low arrival time of jobs in HPC machines we can use CP for online scheduling

Contribution

- Thanks to the precious consideration we propose a complete optimization model for the scheduling and dispatching of real HPC systems.
- The model works in a plug-and-play fashion with one of the most widespread commercial scheduler (PBS Professional).
- Our approach enables a controlled trade-off between schedule computation-time and solution space exploration.

Overview of the approach

At each system event:

Get information on **all** the waiting jobs and on system status

↓

Assign nodes & candidate start times

↓

Dispatch the jobs scheduled at time T

required resources
+ durations

Solve a Allocation & Scheduling problem

The Model

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

1. We set the start time, for all the jobs (jobi) in queue, to be higher than the current time instant
2. We set the start time of all the jobs in execution equal to the start time saved in PBS
3. With the alternative constraint we give the possibility to every job unit (UNijk) of the jobi to be dispatched in one of the nodes of the system
4. With the cumulative constraint we limit the resources utilization to the physical limit

Objective function

CINECA declares to the users an expected waiting (ewti) for each queue of the system:

- Debug queue: 1 hour
- Parallel queue: 5 hours
- Long parallel queue: 24 hours

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

The objective function weight the job waiting on the expected waiting time. In this way all the waitings are fairly distributed in proportion to the expected Waiting of the user.

Others features

Stopped queue:

$$job_i = \perp \;\; \forall i \text{ jobs in a stopped queue}$$

The system administrator can temporarily stop a queue, maintaining the possibility for the user to submit jobs to that queue

Prime-time and Non-prime-time queue:
Prime-time queue can execute only in office hours, non-prime-time queue only in non office hours. To model this feature, we have to obtain a number of intervals sufficient to cover the scheduling horizon:

- We obtain a upper bound of the makespan

$$\lceil MK \rceil = \min_i(s(i)) + \sum_i d_i \;\forall i$$

- We generate all the prime-time (NPIntervals(…)) and non-prime-time (Pintervals(…)) intervals from the current instant e the makespan upperbound, then we constraint prime-time jobs to no overlap non-prime-time intervals and vice versa

$$noOverlap(job_i, NPIntervals(t, \lceil MK \rceil)) \;\forall i \in P$$
$$noOverlap(job_i, PIntervals(t, \lceil MK \rceil)) \;\forall i \in N$$

Others features

Reservations:
A reservation can be seen both as a job and a queue, it require a set of nodes/resources with a given start time (differently from a job) and for a amount of time. When a reservation start the user can submit to it as if it were a queue. For this reason we treat reservations in the main model as jobs but we constraint the start time:

$$job_i = s(i)\ \forall i \text{ reservations}$$

Then we create a new model for each reservation to schedule jobs submitted to it. The reservations job can see only the portion of system of the reservation and they have only a given amount of time to execute:

$$job_i \geq \max(t, s(resv))\ or\ \perp \quad \forall i \in A$$

$$job_i \leq s(resv) + d(resv) - d(a) \quad \forall i \in A$$

$$job_i = s(b) \quad \forall i \in B$$

$$alternative(job_i, UN_{ijk}, u_i)\ \forall i = 1..N$$

$$cumulative(UN_{ijk}, d_i^{P_{ij}}, r_{ijkl}^{P_{ij}}, \sum_{k'} resvUtil_{jk'l})\ \forall k = 1..M, l \in R$$

Others features

Feasibility Check:

In order to avoid user's error on jobs submission we implemented a feasibility check, this check preventively remove jobs and reservations that will lead to an infeasible instance of the model (e.g. a job unit that require more cores than the maximum number of cores present in a node).

We subdivided this check in two step:

1. For each job we create a model to check if the job can be scheduled (we hypothesize that all nodes are running and no other job is in the system):

$$job_1 \geq t \; or \; \bot$$

$$alternative(job_1, UN_{1jk}, u_1)$$

$$cumulative(UN_{1jk}, d_1^{P_{1j}}, r_{1jkl}^{P_{1j}}, rl_{jl}) \; \forall k = 1..M, l \in R$$

2. For each reservation we have to check if it can be scheduled at a given time instant (we have to check if the current running jobs permit this)

$$job_i = s(i) \; \forall i \in A$$

$$job_i = \bot \;\; \forall i \in B$$

$$job_i = s(i) \; \forall i \in S$$

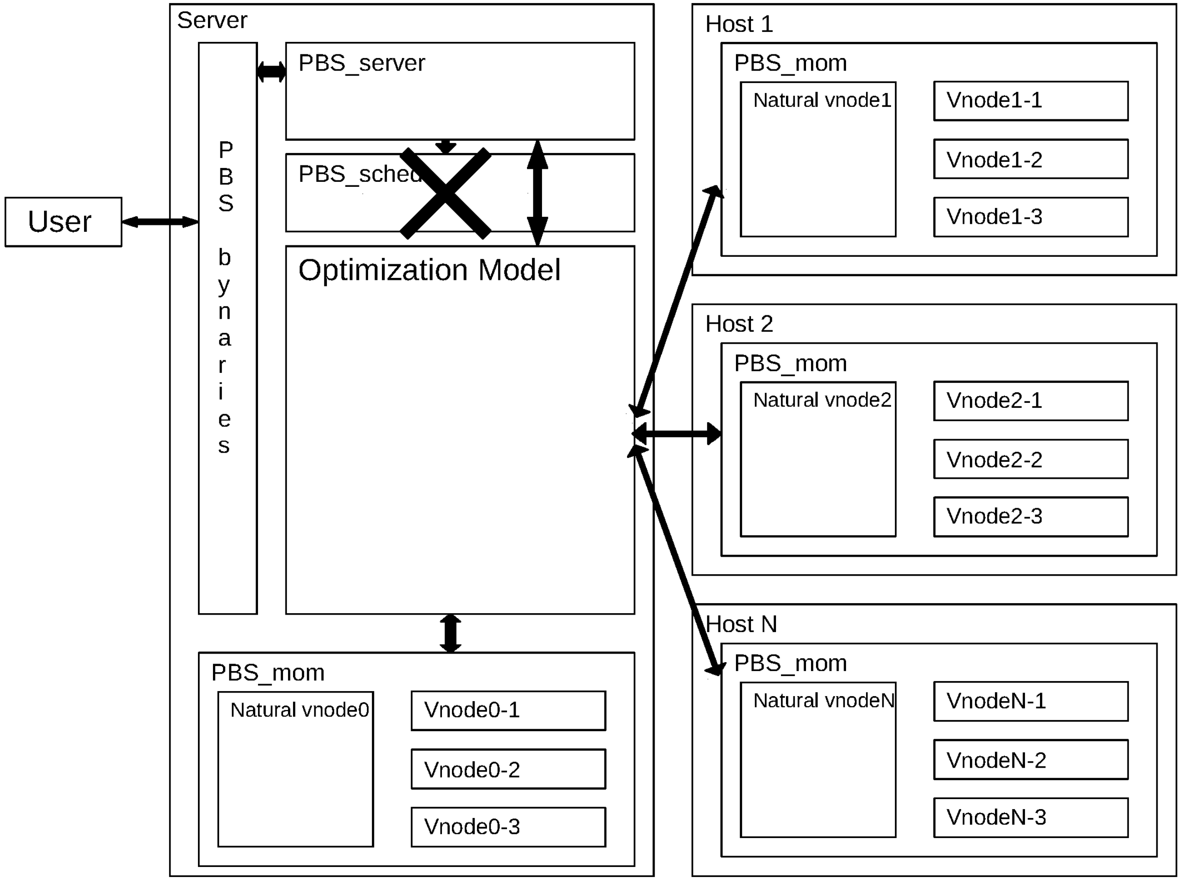
$$job_i = s(i) \; or \; \bot \;\; \forall i \in F$$

$$alternative(job_i, UN_{ijk}, u_i) \; \forall i \in A \bigcup B \bigcup S \bigcup F$$

$$cumulative(UN_{ijk}, d_i^{P_{ij}}, r_{ijkl}^{P_{ij}}, rl_{jl}) \; \forall k = 1..M, l \in R$$

Architecture

The solver work as a plug-in for PBS Professional: PBS Binaries and PBS Server are used for the user interaction, it substitute the PBS Scheduler and PBS Moms are used for the node interaction and job execution



Simulations

Simulated test:

- Synthetic jobs (sleep commands)
- Jobs resources randomly generated from Eurora statistics
- Jobs duration and arrivals randomly generated from Fermi statistics
- Different instances with increasing hardness
- Scheduled compared with two different setup: by PBS with FIFO policy (PBSFifo) and PBS with jobs ordered by walltime (PBSWalltime)

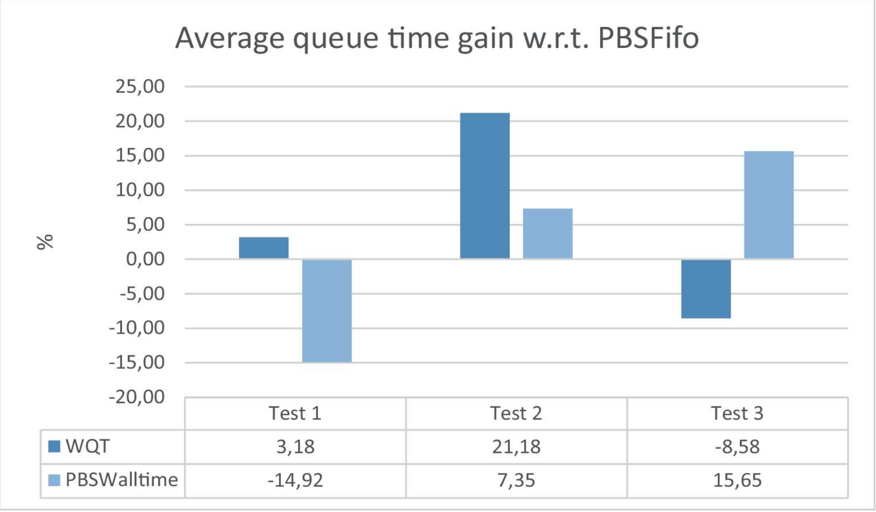
Instances:

- Low hardness: 4 nodes and 99 jobs (Test 1)
- Medium hardness: 65 nodes and 330 jobs (Test 2)
- High hardness: 65 nodes and 700 jobs (Test 3)

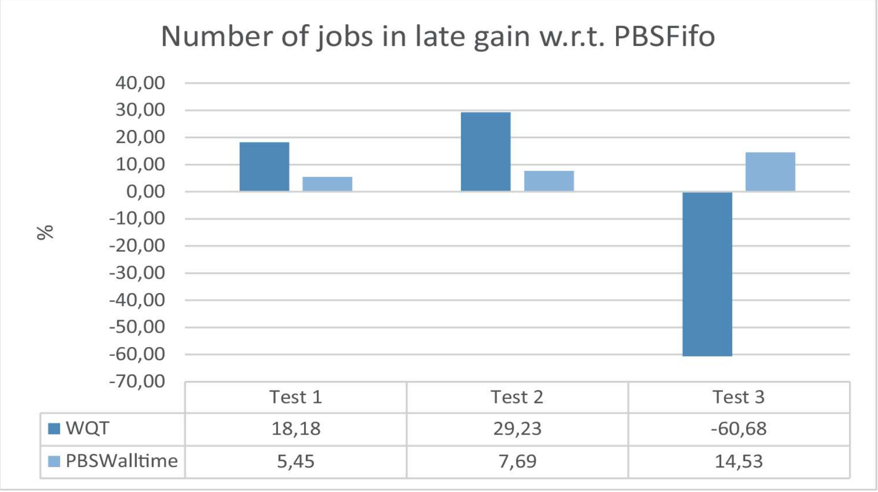
Simulation Results

- Average queue time: Test1 3,18% of improvement, Test2: 21,18% of improvement and Test3 8,58% of worsening

- Number of jobs in late: Test1 18,18% of improvement, Test2 29,23% of improvement and Test3 60,68% worsening



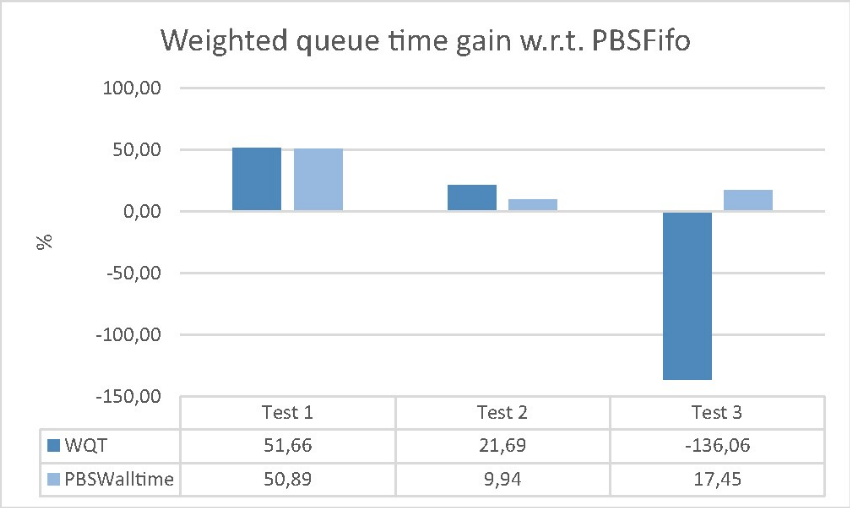
	Test 1	Test 2	Test 3
WQT	3,18	21,18	-8,58
PBSWalltime	-14,92	7,35	15,65

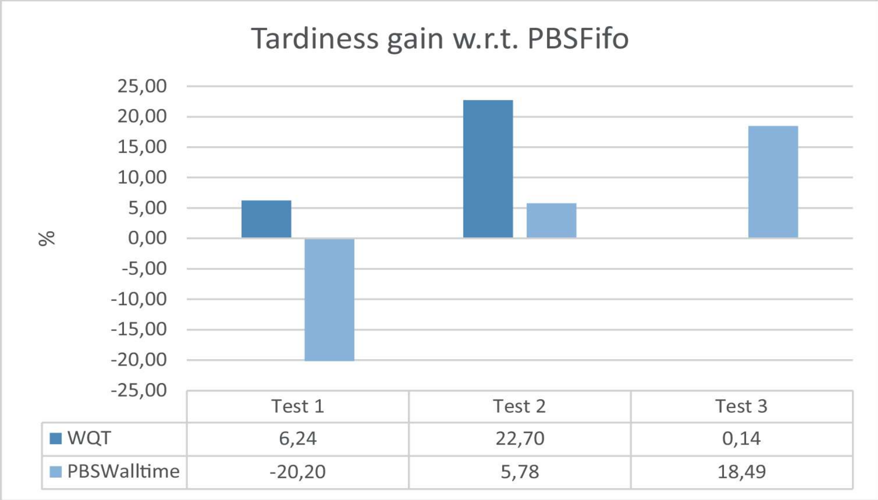


	Test 1	Test 2	Test 3
WQT	18,18	29,23	-60,68
PBSWalltime	5,45	7,69	14,53

Simulation Results

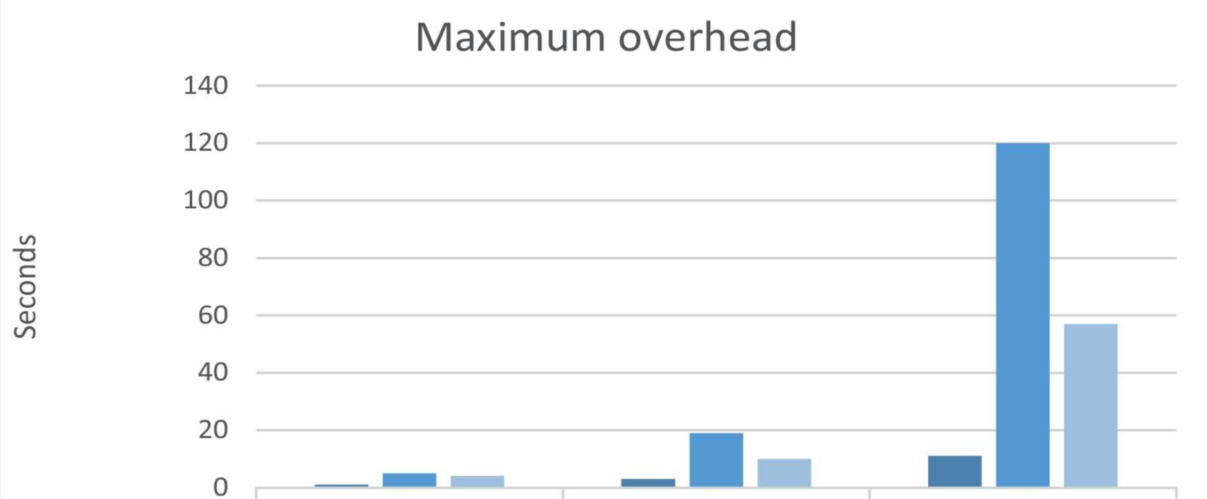
- Weighted queue time: Test1 51,66% of improvement, Test2 21,69% of improvement and Test3 136,06% of worsening
- Tardiness: Test1 6,24% of improvement, Test2 22,70% of improvement and Test3 0,14% of improvement





Simulation Results

- Overhead in general much higher than PBS.
- It does not affect results in medium/low instances
- Increase exponentially

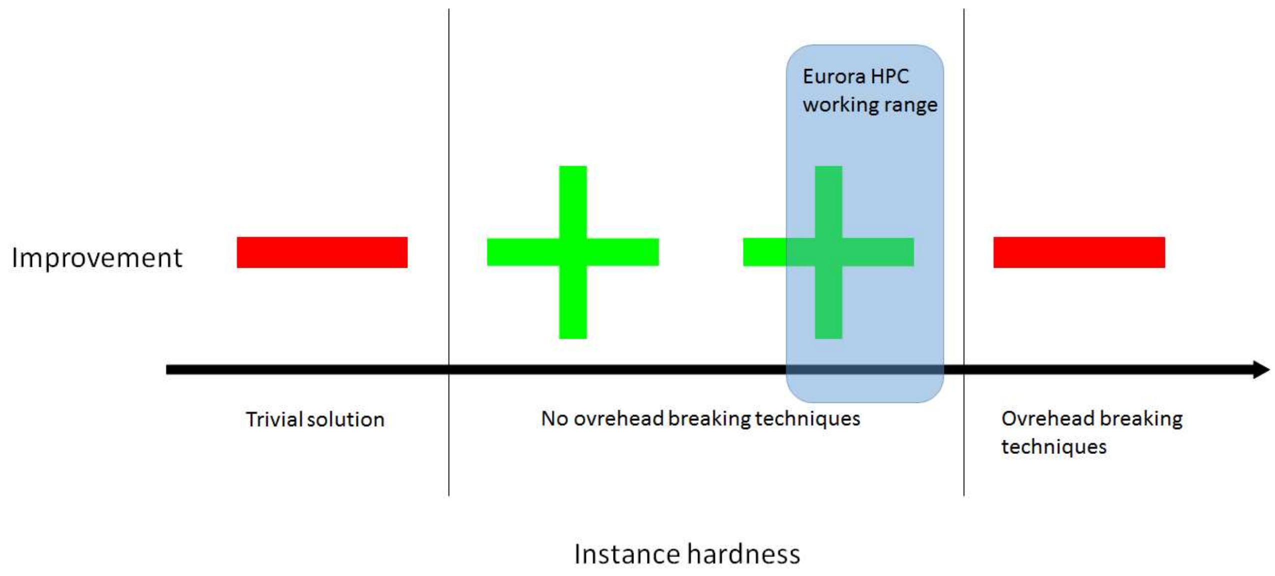


	Test 1	Test 2	Test 3
PBSFifo	1	3	11
WQT	5	19	120
PBSWalltime	4	10	57

Algorithm portfolio selection

Three different ranges:

1. Trivial instances: the overhead due to interaction with PBS lead to a worse solution than PBSFifo
2. Low/Medium instances: High improvements of fair waitings, waitings and lates
3. High instances: the hardness of the instance does not give the possibility to improve the result in an acceptable amount of time



Evaluation on the Eurora HPC

- Evaluation on 5 weeks in a in-production environment
- Users unaware of the testing
- Statistics on different kind of jobs

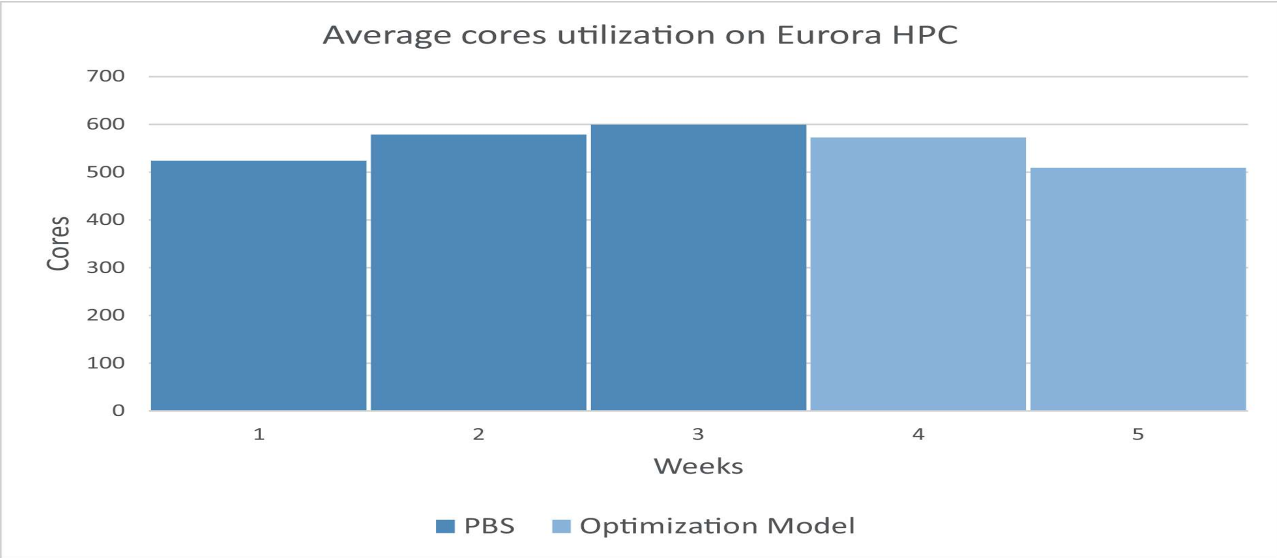
Evaluation Results

- Weighted queue time per job of our model of 2,50*10^-6, PBSFifo 3,93*10^-6 .
- The intervals [Average-δ/2, Average+δ/2] of our CP Scheduler and PBSFifo does not overlap

	WQT	PBS
Average+σ/2	3,32E-006	4,58E-006
Average-σ/2	1,69E-006	3,29E-006
Average	2,50369E-06	3,93E-006

Evaluation Results

- Users were unaware of the new scheduler
- Utilization did not changed significantly



Conclusions

In conclusion:

- We presented a scheduler, based on CP techniques, that can improve the results obtained from commercial schedulers highly tuned for a production environment.
- We implemented all the features to make 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 the proposed solution can be inserted in a portfolio of scheduling algorithms and dominates commercial approaches under instance hardness condition

Future work

- A new prototype under development
- Based on a solver by Google (free for commercial use)
- Improved scalability
- Better overhead-reduction techniques
- Interaction with the cooling system
- Thermal and power management