# A Two-level Scheduler to Dynamically Schedule a Stream of Batch Jobs in Large-scale Grids

Marco Pasquali*
Information Science and Technologies Institute, CNR
Pisa, Italy
m.pasquali@isti.cnr.it

Ranieri Baraglia
Information Science and Technologies Institute, CNR
Pisa, Italy
r.baraglia@isti.cnr.it

Gabriele Capannini
Information Science and Technologies Institute, CNR
Pisa, Italy
g.capannini@isti.cnr.it

Laura Ricci
Computer Science Dept.
University of Pisa
Pisa, Italy
ricci@di.unipi.it

Domenico Laforenza
Information Science and Technologies Institute, CNR
Pisa, Italy
d.laforenza@isti.cnr.it

## ABSTRACT

This paper describes the study conducted to design and evaluate a two-level on-line scheduler to dynamically schedule a stream of sequential and multi-threaded batch jobs on large-scale grids, made up of interconnected clusters of heterogeneous machines. The scheduler aims to schedule arriving jobs respecting their computational and deadline requirements, and optimizing the utilization of hardware resources as well as software resources.

## Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods, Scheduling; F.2.2 [**Nonnumerical Algorithms and Problems**]: Sequencing and scheduling

## General Terms

Algorithms, Experimentation, Management, Performance.

## Keywords

Grids, Scheduling, Simulation.

## 1. INTRODUCTION

To build a grid infrastructure requires the development and deployment of middle-ware, services, and tools. At middle-ware level the scheduler plays a major role in order to efficiently and effectively schedule submitted jobs on the available resources. The objective of the scheduler is to assign tasks to specific resources maximizing the overall resource utilization and guaranteeing the QoS required by applications. In this paper we describe the study conducted to develop a two-level scheduler to dynamically schedule a stream of batch jobs in large-scale grids, made of interconnected clusters of heterogeneous machines. In our study we consider a stream of batch jobs, which arrive to the system and are stored into a single job queue. We suppose that each job is independent of the computation of other jobs, sequential or multi-threaded, that a job is executed only on one machine, and that jobs are allocated to a machine according to the space sharing policy. We also assume that all jobs are not preemptable, and that mechanisms to notify configuration changes, such as job submission/ending, are available in the computing platform.

*Lucca Institute for Advanced Studies, IMT Lucca, Lucca, Italy


*HPDC'08*, June 23–27, 2008, Boston, Massachusetts, USA.
ACM 978-1-59593-997-5/08/06.

## 2. SCHEDULER ARCHITECTURE

At the higher level there is the *Meta Scheduler* (MS), which manages the queue to which users submit their jobs and makes decisions to dispatch them to the lower-level scheduling instances: the *Local Scheduler* (LS). LS makes decisions to schedule jobs on the machines belonging to its cluster. To make scheduling decisions, both MS and LS compute a priority value for each submitted job by exploiting, each one, a different set of heuristics. The computational complexity of the heuristics used at MS level is smaller than those used at LS level.

MS computes the priority $p_j$ of a job $j$ by averaging the results of three heuristics: *Deadline*, which evaluates the average of the differences between available time till the deadline and estimated processing time, computed for a fixed sized set of elements in the rear of the incoming jobs, *Licenses*, which favors the execution of jobs improving the software licenses usage, and *User*, which prioritizes jobs according the user peculiarities. MS dispatches jobs to LSs according to a policy based on two functions: *Load* and *Ordering*. Load aims to balance the workload among clusters. Ordering aims to balance the number of jobs with equal priority in each cluster queue.

For each cluster $c$ two arrays $L_c$ and $O_c$ of $n$ elements are defined, with each array entry corresponding to a priority value, i.e. $p \in [1, n]$. Each $L_c$ stores the amount of workload due to jobs scheduled on $c$ computed as:

$\forall c.\ \forall p \in [1, n].\ L_c[p] = \sum_{\{j \in c | priority(j) \geq p\}} workload(j)$

Load assigns a job $j$ to the first empty cluster $\bar{c}$ (i.e. $L_{\bar{c}}[p_j] = 0$) or to the one with the least workload corresponding to $p_j$ (i.e. $L_{\bar{c}}[p_j] = min\{L_c[p_j]\}_{\forall c}$). If exists more than

| *cluster* | *optimal* | $t_a = 0$ | $t_a = 5$ | $t_a = 10$ | $t_a = 15$ |
|---|---|---|---|---|---|
| $c_1$ | 0.52 | 0.25 | 0.53 | 0.63 | 0.67 |
| $c_2$ | 0.26 | 0.25 | 0.23 | 0.24 | 0.25 |
| $c_3$ | 0.13 | 0.25 | 0.14 | 0.09 | 0.07 |
| $c_4$ | 0.07 | 0.25 | 0.08 | 0.05 | 0.01 |

**Table 1: MS's workload result.**

one cluster with the same workload, Ordering is invoked. It works using the same strategy of Loads, but uses $O_c$ arrays to store for each priority how many jobs are queued in the cluster $c$.

As LS we experimented a Flexible Backfilling [2] and an EASY Backfilling algorithm [3]. In the case of Flexible Backfilling, the job priorities were computed as the sum of the weighted contributes computed by four heuristics obtained from the ones in [1, 2]: *Aging*, which increases its result for each job proportionally to the time elapsed since its submission, *Deadline*, which favors the jobs closer to their deadline, *License*, which assigns a higher increase to jobs requiring a larger number of critical licenses, and *Wait Minimization*, which tries to improve the total flow time favors jobs with a shorter estimated execution time.

## 3. PERFORMANCE EVALUATION

The objective of the executed tests is to investigate about workload distribution and job classification choices made by the MS scheduling policies we propose.

Tests were conducted by using a synthetic workload (5000 jobs, 20 licenses, and with the 30% of jobs without deadline), and simulating a grid consisting of four clusters (from $c_1$ to $c_4$ in Table 1) each one with a different number of machines, i.e. 120, 60, 30, and 15 respectively. In order to obtain stable values, each simulation was repeated 20 times with different streams. Table 1 shows the average percentage of total workload assigned to each cluster. Increasing the average job interarrival time ($t_a$), it happens that some clusters are enough powerful to maintains empty their LS's queue. Consequently, MS dispatches a larger number of jobs to such clusters. $t_a = 5$ obtains a workload distribution that better approximates the optimal one. It is because of the amount of workload assigned to LSs, properly represents the clusters computational power.

Consequently, to figure out the quality of the MS's job classification, we show the results obtained by using $t_a = 5$. Three different cases were evaluated: in the first one, referred as $h(ms)$, the Flexible Backfilling algorithm is used as LS exploiting job priorities computed by MS, in the second one, referred as $h(ls)$, at LS level, is still applied the Flexible Backfilling algorithm exploiting its job classification heuristics, in the third case, referred as $fcfs$, jobs are scheduled at both MS and LS level according to the FCFS policy. Figure 1 shows the average slowdown[2] computed for all the jobs without deadlines. It can be seen that it improves in inverse proportion to the cluster computational power. This because the MS objective is to maintain the same workload in each cluster queue. Consequently, it dispatches jobs among clusters without considering the time a job spent in the queue. Figure 1 shows also the percentage of jobs ended after their deadlines (referred as delayed job). It can be seen that the solutions based on LS and MS heuristics are able to improve the number of jobs executed

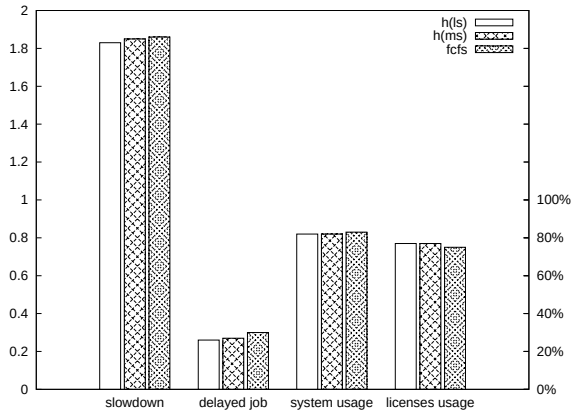


**Figure 1: MS's job classification result.**

within their deadline, compared to the EASY Backfilling solution. Furthermore, considering system and licenses usage, it can be seen that using only the MS job classification we obtained results comparable to those obtained by using the LS one, but with a smaller computational cost.

## 4. CONCLUSIONS

This paper describes the study we conducted to develop a two-level scheduler to dynamically schedule a stream of batch jobs on grids made up of heterogeneous machines in interconnected clusters. The conducted simulation demonstrated that the proposed solution can be a viable one. In particular, we show that using a lightweight component like MS, joined with lightening LSs, carries out good results as using more complex LSs.

## 5. ACKNOWLEDGMENTS

This work has been supported by SUN Microsystems's grant, and by the European CoreGRID NoE (European Research Network on Foundations, Software Infrastructures and Applications for Large Scale, Distributed, GRID and Peer-to-Peer Technologies, contract no. IST-2002-004265).

## 6. REFERENCES

[1] G. Capannini, R. Baraglia, D. Puppin, L. Ricci, and M. Pasquali. A job scheduling framework for large computing farms. In *SC07*, Reno, USA, November 2007.

[2] A. D.Techiouba, G. Capannini, R. Baraglia, D. Puppin, M. Pasquali, and L. Ricci. Backfilling strategies for scheduling streams of jobs on computational farms. In *CoreGRID Workshop*, Crete, Greece, June 2007.

[3] A. W. Muálem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel and Distributed Syst.*, 12(6), June 2001.