Including accurate user estimates in HPC schedulers: an empirical analysis

Nestor Rocchetti, Santiago Iturriaga, and Sergio Nesmachnow

[1] Universidad de la República
Montevideo, Uruguay
{nrocchetti, siturria, sergion}@fing.edu.uy

Abstract. This article focuses on the problem of dealing with low accuracy of job runtime estimates provided by users of high performance computing systems. The main goal of the study is to evaluate the benefits on the system utilization of providing accurate estimations, in order to motivate users to make an effort to provide better estimates. We propose the Penalty Scheduling Policy for including information about user estimates. The experimental evaluation is performed over realistic workload and scenarios, and validated by the use of a job scheduler simulator. We simulated different static and dynamic scenarios, which emulate diverse user behavior regarding the estimation of jobs runtime. Results demonstrate that the accuracy of users runtime estimates influences the waiting time of jobs. Under our proposed policy, in a scenario where users improve their estimates, waiting time of users with high accuracy can be up to *2.43 times* lower than users with the lowest accuracy.

Keywords: high performance computing, scheduling, execution time estimation, quality of service.

1 Introduction

Parallel supercomputers are high-end machines designed to support the execution of parallel jobs [1]. Nowadays, supercomputers have become a common commodity in scientific oriented companies and research institutions, especially those working on High Performance Computing (HPC). Along with the development of HPC infrastructures, the main trend has been using commercial cluster management software suites. These software suites offer a wide variety of features, which include queue management, process prioritization, and scheduling algorithms [2].

Due to the increasing usage of supercomputers, job scheduling has become a critical task, where small differences in policies can result in great changes in resource utilization, and in performance [3]. The most popular scheduling policy used in batch schedulers is first-come, first-served (FCFS) [2]. This scheduling policy often comes in combination with a backfilling method called EASY-Backfilling. The idea of this method is to select small jobs (i.e., jobs with low number of requested cores or walltime) for execution before the time they were supposed to, whenever holes of idle resources appear [4]. Backfill systems relay on users job runtime estimates to accomplish their task.

Execution time estimation has a significant impact on how a scheduler treats different jobs, and on general performance [4]. The inaccuracy of user estimates worsens the overall performance of the parallel system [5]. For this reason, many studies have been performed in order to improve runtime estimates, to make a positive impact on both system-related and user-related performance metrics.

This article focuses on the problem of dealing with low accuracy of job runtime estimates provided by users. The main goal of the study is to evaluate the benefits of providing accurate estimation, in order to motivate users to make an effort to better estimate the system utilization.

The main contributions of this article are: i) the study of the impact of user runtime estimations in the system utilization for current HPC infrastructures; ii) the design and implementation of a novel scheduling strategy, named *Penalty Scheduling Policy* (PSP), which prioritizes jobs from users that provide good estimates on jobs runtime; and iii) the experimental evaluation of PSP using realistic workloads, on both static and dynamic scenarios, which emulate diverse user behavior regarding the estimation of jobs runtime.

We present an empirical evaluation of PSP under five scenarios that represent different user behaviors regarding the estimated runtime of jobs. For each scenario, four simulations are performed considering different workload patterns that models the real situation of our HPC infrastructure, Cluster FING. Then, we analyze the impact of their accuracy on the queuing time of their jobs.

The paper is organized as follows. Section 2 introduces some general concepts about scheduling. A review of related work is presented in Section 3. Section 4 describes the proposed PSP algorithm. Section 5 presents the workload analysis and the problem instances characteristics. Then, the experimental evaluation of PSP is presented in section 6. Finally, section 7 presents the conclusions and formulates the main lines for future work.

2 Background

This section presents a brief description of Cluster FING at Facultad de Ingeniería and the SLURM simulator [6], the tool used to perform the scheduling evaluation.

2.1 Cluster FING

Cluster description. Cluster FING [7] is the HPC infrastructure at Facultad de Ingeniería, Universidad de la República, Uruguay. It is an heterogeneous cluster of computing resources with 1672 cores, which has been operational since 2008, with a steady growth in components. It is used mostly for the batch execution of scientific and engineering computing jobs.

Job scheduling. Cluster FING uses Maui [8] for job administration. Maui is a policy engine to manage resources (such as processors, memory, and disk) that are assigned to jobs. It also provides other features like mechanisms for resource usage optimization, monitor system performance, help diagnose problems, and general system manage. The default behavior of Maui is defined by a first-come, first-served (FCFS) batch scheduler, plus EASY Backfilling [8].

FCFS is a queue policy where the jobs are attended in the same order that they arrive: the first job to arrive is the first to get access to the requested resources. Backfilling is a policy that requires users to estimate the runtime of their jobs. Provided this information of runtime, short (runtime) jobs are allowed to execute before a larger job at front of the queue [9]. The EASY Backfilling algorithm only moves ahead jobs that do not delay the job at the head of the queue.

2.2 The SLURM workload manger

SLURM (Simple Linux Utility for Resource Management) [10] is an open-source workload manager designed for clusters running Linux. SLURM provides the basic workload manager tasks for allocating resources to users for a requested amount of time. It also provides tools for starting, executing, and monitoring jobs on a set of allocated nodes. Besides that, it manages a queue of pending work that is configured by the administrators of the application/infrastructure.

SLURM design is modular, including many optional built-in plugins. Two relevant plugins that are used in this work are *SLURM Priority Plugin API* and *SLURM Accounting Storage Plugin API*. SLURM Priority Plugin API allows computing the priority of the queued jobs in every iteration. The default configuration of this plugin is the basic implementation, which provides a basic FIFO job priority. It also comes with a multifactor job priority plugin that can be configured easily. SLURM Accounting Storage Plugin API allows the storage of accounting data collected during the execution of the scheduler, it can be configured to use a MySQL database in order to store accounting data for future processing. We use SLURM Accounting Storage Plugin API to store the accounting data in the simulations performed to evaluate the priority scheduler considering user runtime estimates proposed in this work.

In this work, we have adapted SLURM Priority Plugin API to implement our proposed priority policy. It is important to state that, in SLURM, the larger the priority number, the higher the job will be positioned in the queue, and the sooner the job will be executed.

2.3 The SLURM simulator

The SLURM simulator [6] is a job trace simulator that uses the SLURM scheduler as the simulation tool with minor SLURM code changes. The implementation of the simulator was left outside the SLURM source code; this way the simulation mode can be used with future releases of the scheduler. The simulator contains two programs, external to SLURM: *sim_mgr*, the simulation manager, which keeps control of the simulation time and *sim_lib*, the simulation library, which captures time-related calls and synchronizes with *sim_mgr* for sleep calls or getting simulation time.

A workload generator for SLURM is provided with the simulator. This workload generator, with slight changes in its source code, is used in this article to create the synthetic workloads used in the experimental evaluation of the proposed scheduler. The workload generated is based on real workload registered on Cluster FING. The hardware infrastructure used on the simulations is also based on Cluster FING (see details about the problem instances on Section 5).

3 Related work

This section describes the related work about analyzing user runtime estimates, its impact on job scheduling, and proposed techniques to improve the accuracy of the estimations.

Several relevant related works reported that user runtime estimates of jobs are usually inaccurate. For example, Cirne and Berman [1] showed that in four traces of different supercomputers, 50% to 60% of jobs made use of less than 20% of their requested time. Other features were also reported, for example the relation between failed jobs and accuracy, and between job length and accuracy.

The impact of user runtime estimates has been a matter of study in many articles. As stated by Tsafrir [5], some of the studies performed gave surprising, counterintuitive results. While some researchers found that inaccurate estimates are usually preferable over accurate ones, other studies show that performance is insensitive to accuracy of users runtime estimates [3,11 −14]. Tsafrir reported results showing that performance is affected by the quality of users runtime estimates.

The empirical study by Tang et al. [3] showed that FCFS is not sensitive to user runtime estimates. However, using accurate runtime estimates improve performance on scheduling policies that give precedence to short jobs, like Shortest Job First. It is also presented a scheme that uses historical information about the quality of estimates of both user and project scopes, to redefine the runtime estimate of a given job. The proposed adjusting scheme is transparent to users and easy to deploy.

In Iturriaga et al. [15], we studied the problem of energy consumption in heterogeneous computing scenarios proposing novel scheduling algorithms and reporting their experimental evaluation performed over realistic workloads and scenarios. We analyzed three real-world task workloads and proposed a workload generation model considering uncertainties. We computed improvements of up to 32% in computing performance and up to 18% in energy consumption.

In this line of work, this article focuses on analyzing the impact of users improving their runtime estimates when using the proposed PSP in HPC clusters. PSP is based on lowering the priority of jobs submitted by users whose runtime estimates have been inaccurate in the past, as it is described in the following section.

4 The proposed Penalty Scheduling Policy

The penalty policy applied in PSP consists in affecting the priority of jobs according to the historical precision of runtime estimates of the users.

We define the accuracy of a users job runtime estimate as $A = \frac{t_{run}}{t_{req}}$, where t_{run} is the real runtime of the job, and t_{req} is the requested time. Accuracy can take values between 0.0 and 1.0, thus the average accuracy is also between that interval. The bigger the average accuracy, the better the user is when estimating runtime, and the PSP method will assign higher priority to the users newly submitted jobs.

To affect the priority in the PSP scheduler, the accuracy of users estimates is used. We used a dynamic update scheme for estimating the accuracy of users, by computing the average deviations (i.e., ratio) between estimated time and real execution time for the last ten completed jobs for each user.

Table 1 shows the intervals used to assign priority to jobs. The priority is a number between 1 and 5, a higher number means that the jobs is closer to the head of the queue. For example, a job whose user has an accuracy of 0.35 will have a priority of 2. This priority is first calculated when the job is submitted, and it is updated every time a new job is submitted or when releasing resources (i.e., a job ends).

Table 1. Intervals for accuracy of estimates and priorities for each tag names.

tag name	*accuracy interval*	*priority*
a1	[0.0,0.2)	1
a2	[0.2,0.4)	2
a3	[0.4,0.6)	3
a4	[0.6,0.8)	4
a5	[0.8,1.0]	5

We consider that a job runtime estimate is "good" when its accuracy is over 0.6, under that it is considered a poor quality estimate. That consideration is based on the study of workload trace at Cluster FING, in which the users with coefficient of accuracy of estimates higher than 0.6 is just 4% of the total platform users.

Algorithm 1 presents a pseudocode for the implementation of the proposed scheduler into SLURM.

Algorithm 1. PSP implementation in SLURM

```
priority_thread_tasks()
  while (true)
    waitEvent(job_completion, job_submission, time_lap, ...);
    jobs_list.computeNewPriority();
  end;
end;
scheduling_thread_tasks()
  while(true)
  //scheduling thread tasks
  end;
end;
main()
…
scheduling_thread.create();
priority_thread.create();
…
end;
```

We included our code in the Multifactor implementation of SLURM Scheduler Priority Plugin API. This priority API is used by the *Job Manager,* which is the component that accepts jobs requests and includes pending jobs in a priority ordered queue. The function computeNewPriority() called by the priority_thread communicates with that API and updates the priority of all jobs in pending state based on data retrieved from the database in which job accounting information is stored. This function is called periodically and when there is a change in a job state that may permit another job to begin execution.

5 Workload analysis and problem instances

The design of realistic problem instances is a very relevant issue when dealing with the evaluation the new approaches for scheduling and managing HPC infrastructures. We analyzed the workload of Cluster FING in order to gather real information for creating realistic instances of the scheduling problem (including workloads and user behavior when estimating jobs runtime). This section summarizes the main findings about workload analysis and users job runtime estimates and describes the problem instances generated.

5.1 Workload analysis

We analyzed the complete trace of jobs submitted to Cluster FING between April 2010 and March 2015, containing a total of 276803 jobs. As the main results of the statistical analysis of jobs, we found that almost half (49.3%) were small jobs, with less than a minute of execution time, sequential jobs were 44.2%, and parallel jobs were 6.5%. We found a predominance of power of two number of cores requested in parallel jobs (85.1%).

We computed the average accuracy of users runtime estimates by applying the model described in the previous section. Regarding this average, we divided the users in six groups (a bigger group number means a higher accuracy of estimates): *g1*–0 to 0.05, *g2*–0.05 to 0.15, *g3*–0.15 to 0.25, *g4*–0.25 to 0.35, *g5*–0.35 to 0.60, and *g6*–0.60 to 1.0. These groups have 21%, 21%, 18%, 17%, 19%, and 4% of the 117 regular users of the cluster respectively.

5.2 Problem instances

Using the information gathered in the workload analysis, we created problem instances to evaluate the PSP scheduling algorithm under different scenarios.

The simulated infrastructure consists of 37 machines with 12 cores each (a total number of 444 cores). We also configured an execution queue that accepts serial, and parallel jobs requesting up to 16 cores, and up to ten days of execution time. Regarding the task workload generation, we used the software included with the SLURM simulator, and customized its source code to generate specific instances for the problem to study. The changes include adding constraints on the number of cores

requested and maximum requested job runtime, so the jobs generated fulfill the constraints of the execution queue configured.
Each generated workload has 1000 jobs, from 20 users. Each job demands a number of cores that is a power of two between 1 to 16, and up to 10 days of runtime execution. The distribution of the number of cores and runtime execution is representative of the workload at Cluster FING.

In order to test different accuracy of estimates situations, six scenarios simulating different user behavior were generated, the main characteristics of those scenarios are shown on Table 2. Four scenarios are *static* regarding the accuracy on runtime estimates, while the other two emulate users that learn and improve the accuracy of their execution time estimation.

Table 2. Scenarios generated to simulate different user behavior.

scenario	*type*	*accuracy*	*learning schema*
BE	static	group a1 (0.0–0.2)	none
GE	static	group a6 (0.8–1.0)	none
CF	static	groups g1 to g6	none
CFG	static	groups g1 to g6	none
DI	dynamic	incremental improving	all users
HDI	dynamic	incremental improving	half of the users

The first scenario is BE (*Bad Estimates*), in which all users have the worst level of job runtime estimate accuracy (group a1, defined in Section 4), with accuracy between 0.0 and 0.2. In the second scenario, GE (*Good Estimates*), every user has the highest level of accuracy of job runtime estimates (group a6), with an accuracy between 0.8 and 1.0. The third scenario is CF (Cluster FING), where the users accuracy was generated so it is representative of the one accounted at Cluster FING. We divided the accuracy in the six groups, g1 to g6, defined in the previous subsection.

The fourth scenario is CFG (Cluster FING with good estimates), introducing changes in CF scenario to model a situation where almost half the user estimations are in group a6. In order to keep coherence we changed the weight of the groups as follows: a1: 12%, a2: 12%, a3: 9%, a4: 8%, a5: 10%, and a6: 49%.

The last two are dynamic scenarios, which emulate a rational behavior of users that gradually learn how to estimate job execution times. The improvement of estimations is calculated with a frequency of 5 jobs (i.e. every 5 jobs submitted by the user) as follows: if $accuracy \leq 0.5$, then it is increased by $(1-accuracy)\times 0.1$; else $accuracy$ is increased by $accuracy\times 0.1$. In the fifth scenario, DI (Dynamic Improvement), all users improve their estimations as described. On the other hand, in the last scenario HDI (Half Dynamic Improvement), only half of the users were modeled to correctly learn how to improvement on their estimates.

The source code of the SLURM synthetic workload generator was modified in order to emulate this six different scenarios of user behavior, and generate them at once, using the exact same workload trace.

6 Experimental analysis

This section reports the experimental analysis of the proposed PSP algorithm over 24 scenarios defined from the combinations of infrastructure, workload, and estimations. All simulations were performed in a virtual machine running Ubuntu v14.04. The results of each simulation were stored in a MySQL database, a functionality provided by SLURM scheduler.

In order to get information about each class of accuracy of users job runtime estimates, we studied the average waiting time per user for the five classes. Table 3 reports, for each of the six scenarios evaluated, the average waiting time (in minutes) of each of the accuracy classes defined. We identify the empty classes with a "–".

Table 3. Average waiting time for each scenario divided by accuracy class.

scenario	***average waiting time (minutes)***				
	a1	***a2***	***a3***	***a4***	***a5***
BE	820.05	–	–	–	–
GE	–	–	–	–	796.18
CF	1093.12	637.53	519.30	450.30	–
CFG	1488.60	986.33	759.15	706.35	517.35
DI	–	1030.25	804.93	534.15	424.43
HDI	1091.58	837.51	781.38	509.70	484.20

We compared the waiting time for users that did not increase their accuracy, and the users that did. First we discuss the results of static scenarios (CF and CFG), then we continue with dynamic scenarios (DI and HDI).

CF is a static scenario in which the highest accuracy class with users is *a4* (0.6 to 0.8). This class has an average waiting time of *450.3 minutes,* which is *2.43, 1.82, and 1.77 times* lower than the average waiting time of class a1, BE, and GE respectively. Only one user was on class *a4,* and his jobs were of the highest priority on the simulation.

Scenario CFG is quite different from CF: there are 6 users on the higher class (*a5*). The average waiting time per user in a5 is *517.35 minutes*, which is *2.88, 1.59,* and *1.54 times* lower than the average waiting time of class a1, BE, and GE respectively. The waiting time in CFG is also higher than the one in CF; the reason is that more users are on class a5, and they compete with each other for computing resources.

DI and HDI are dynamic scenarios, with improvements on the accuracy of users runtime estimates. Due to the dynamism, we decided to include the average waiting time of each user in the class where he belongs *at the end of the simulation*. In DI, the highest accuracy class is *a5*, having an average waiting time of *424.23 minutes*, which is *2.43*, *1.93*, and *1.88 times* lower than the average waiting time of class a2, BE, and GE respectively. The highest accuracy class in scenario HDI is *a5*, in which the average waiting time is *484.2 minutes*, this value is *2.25*, *1.69*, and *1.64* times lower than the average waiting time of class a1, BE, and GE respectively.

Scenario DI has the lower waiting time of all 6 scenarios for its highest populated class, which is *a5*. DI also has an overall average waiting time of 698.44 minutes, lower than the overall average waiting time for HDI (740.87 minutes). In a scenario where all users improve their estimates, the waiting time improves for all users, even for the ones belonging to the higher class.

Fig. 1 summarizes the average waiting time (in minutes) for jobs submitted for the scenarios CF, CFG, DI, and HDI. The average waiting times are grouped by class, from a1 to a5. For a particular scenario, results show that the waiting time for a job significantly reduces when moving to a higher accuracy class. In a scenario where all users improve their accuracy of runtime estimates, every class experience a drop in their average waiting time, which means a general improvement in the quality of service of the HPC infrastructure. For example, class a2 in DI (the lowest class of the scenario) has lower waiting time than class a1 in all other scenarios in the figure.

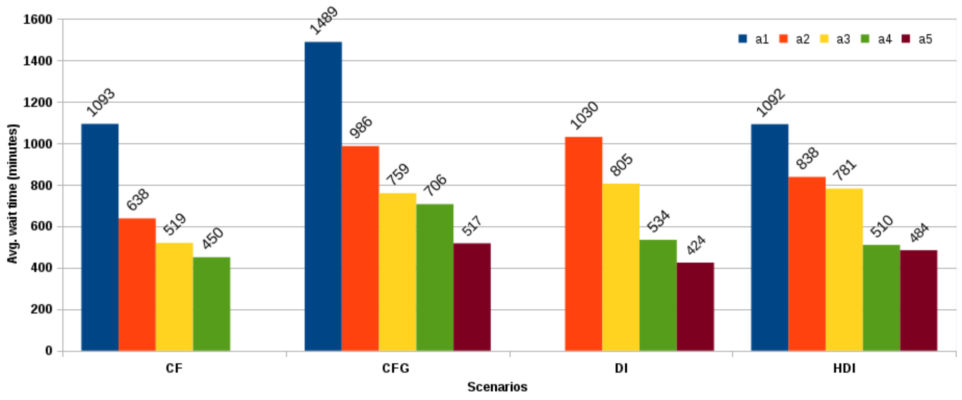

Fig. 1. Average waiting time of each accuracy group, grouped by scenario.

7 Conclusions and future work

This article presented a scheduling policy called *Penalty Scheduling Policy (PSP),* which focuses on the problem of dealing with low accuracy of job runtime estimates provided by users. The goals of the study are twofold. First, promoting users to accurately estimate the execution time of their jobs. Second, to evaluate the benefits of including a policy for a wise planning of computing resources by prioritizing requests from those users that provide accurate estimate for job execution times.

An experimental evaluation of the use of Penalty Scheduling Policy in a simulated computer system environment, developed using the SLURM simulator, was presented. The empirical study analyzed the PSP performance on six different scenarios regarding the user behavior when estimating job time execution. The main results of the experimental analysis show that in an environment where all users improve their estimates, every users experience a improvement on their quality of service. The proposed strategy was included in SLURM, but it can be easily included in other popular resource management systems such as Maui.

The main lines for future work are related to extend the experimental evaluation of the proposed scheduler using different workloads and statistics. The performance of PSP could be studied over synthetic workloads, created with the generator implemented by Tsafrir [5] using realistic (modal) job runtime estimates. This study will help to evaluate the real difference between the impact of bad user job runtime estimates, and good ones. We also plan to test the PSP method in a real environment, for example on Cluster FING, in order to test users acceptance of this policy.

8 References

1. Cirne, W., Berman, F.: A comprehensive model of the supercomputer workload. IEEE International Workshop on Workload Characterization, pp. 140–148 (2001).
2. Etsion, Y., Tsafrir, D.: A short survey of commercial cluster batch schedulers. Technical Report 2005-13. School of Computer Science and Engineering, The Hebrew University of Jerusalem (2005).
3. Tang, W., Desai, N., Buettner, D., Lan, Z.: Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P. IEEE International Symposium on Parallel & Distributed Processing, pp. 1–11 (2010).
4. Tsafrir, D., Etsion, Y., Feitelson, D.: Modeling user runtime estimates. In: 11th international conference on Job Scheduling Strategies for Parallel Processing, pp. 1–35 (2005).
5. Tsafrir, D.: Using inaccurate estimates accurately. In: 15th international conference on Job Scheduling Strategies for Parallel Processing, pp. 208-221 (2010).
6. Lucero, A.: Simulation of batch scheduling using real production-ready software tools. In: 5th Iberian Grid Infrastructure Conference, pp. 345–356 (2011).
7. Nesmachnow, S. Computación científica de alto desempeño en la Facultad de Ingeniería, Universidad de la República, Revista de la Asociación de Ingenieros del Uruguay 61:12–15, 2010 (text in Spanish).
8. Jackson, D., Snell, Q., Clement, M.: Core algorithms of the Maui scheduler. In: 7th international conference on Job Scheduling Strategies for Parallel Processing, pp. 87–102 (2001).
9. Mu'alem, A. W., Feitelson, D. G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Transactions on Parallel and Distributed Systems 12(6):529–543, 2001.
10. Yoo, A. B., Morris, A. J., Grondona, M.: Slurm: Simple linux utility for resource management. In: 9th international conference on Job Scheduling Strategies for Parallel Processing, pp 44–60 (2003).
11. Zotkin, D., Keleher, P. J.: Job-length estimation and performance in backfilling schedulers. In: 8th International Symposium on High Performance Distributed Computing, pp. 236–243 (1999).
12. Zhang, Y., Franke, H., Moreira, J., Sivasubramaniam, A.: Improving parallel job scheduling by combining gang scheduling and backfilling techniques. In: 14th IEEE International Parallel and Distributed Processing Symposium, pp. 133–142 (2000).
13. England, D., Weissman, J., Sadago-pan, J. : A new metric for robustness with application to job scheduling. In: 14th IEEE International Symposium on High Performance Distributed Computing, pp. 135–143 (2005).
14. Guim, F., Corbalán, J., Labarta, J.: Prediction f based models for evaluating backfilling scheduling policies. In: 8th IEEE International Conference on Parallel and Distributed Computing, Applications & Technologies, pp. 9–17 (2007).
15. Iturriaga, S., García, S., Nesmachnow, S.: An Empirical Study of the Robustness of Energy-Aware Schedulers for High Performance Computing Systems under Uncertainty. In High Performance Computing, pp. 143-157 (2014).