Penalty Scheduling Policy Applying User Estimates and Aging for Supercomputing Centers

Nestor Rocchetti[1], Miguel Da Silva[1], Sergio Nesmachnow[1]
Andrei Tchernykh[2]

[1] Universidad de la República,
Julio Herrera y Reissig 565, Montevideo 11300, Uruguay
{nrocchetti, mdasilva, sergion}@fing.edu.uy

[2] CICESE Research Center
Carretera Ensenada-Tijuana 3918, Zona Playitas, Ensenada 22860, B.C. Mexico
chernykh@cicese.mx

Abstract. In this article we address the problem of scheduling on realistic high performance computing facilities using incomplete information about tasks execution times. We introduce a variation of our previous Penalty Scheduling Policy, including an aging scheme that increases the priority of jobs over time. User-provided runtime estimates are applied as in the original Penalty Scheduling Policy, but a realistic priority schema is proposed to avoid starvation. The experimental evaluation of the proposed scheduler is performed using real workload logs, and validated using a job scheduler simulator. We study different realistic workload scenarios to evaluate the performance of the Penalty Scheduling Policy with aging. The main results suggest that using the proposed scheduler with the aging scheme, the waiting time of jobs in the high performance computing facility is significantly reduced (up to 50% in average).

Keywords: high performance computing, scheduling, execution time estimation, aging scheme, Penalty Scheduling Policy.

1 Introduction

Job scheduling has become a critical issue in supercomputing. When dealing with large and complex problems, small differences in scheduling policies can result in great improvements in resource utilization, performance, and energy consumption [1]. One of the most popular scheduling policies is first-come, first-served (FCFS), which is often used in combination with the EASY-Backfilling method [2]. Backfilling systems rely on job runtime estimates provided by users to accomplish the task planning.

Scheduling strategies based on estimations provided by users are popular. However, the inaccuracy of user estimates impacts on the general performance of the system, worsening its overall performance [3]. Some studies in the literature have proposed strategies to improve runtime estimates to help improve the overall performance of High Performance Computing (HPC) systems [4, 5].

In our previous work [4] we introduced the Penalty Scheduling Policy (PSP) for supercomputing centers. PSP consists of assigning a priority to a recently submitted job. This priority is assigned according to the historical accuracy of job runtime estimations from the task owner, computed in a given window of previously completed jobs. The priority is assigned from a set of five priority groups, in which the priority goes from 1 to 5. In that previous work, we studied the performance of PSP with automatically generated workload logs.

In this article, we contribute with a new version of the PSP scheduler proposed on [4]. We also contributed with an extended analysis of the impact of user runtime estimates on the system utilization in HPC infrastructures using real parallel workload data and simulations.

The new version of PSP presented in this article (PSP+AGING) includes a new approach for defining the intervals of accuracy, and an aging scheme that increases the priority of jobs over time to prevent starvation. Finally, we report results on the performance of PSP under scenarios based on workload logs obtained from the Parallel Workload Archive (PWA) [6] and compared with an experimental evaluation of these scenarios using a traditional FCFS scheduling policy.

The paper is organized as follows. Next section presents a review of related work about scheduling using runtime estimates in supercomputing centers. Section 3 describes the proposed PSP+AGING algorithm. Section 4 introduces the workload analysis and the main features of the problem instances considered in the study. Then, the experimental evaluation of PSP+AGING over realistic HPC scenarios is presented in Section 5. Finally, Section 6 presents the conclusions and formulates the main lines for future work.

2 Related work

In this section, we review the related work about analysis of user runtime estimates on parallel supercomputers, its impact on job scheduling, and proposed job scheduling techniques.

According to the seminal work by Lee et al., "it is a well-documented fact that user-provided runtime estimates are inaccurate" [5]. In that article, the authors reviewed the results of previous studies by Cirne and Berman [1], Mahood and West [7], and Chiang, Arpaci-Dusseau, and Vernon [8], in which the previous statement was confirmed. Lee et al. conducted an experiment in which they asked users to estimate their jobs runtime the best they could, and also asked to rate their confidence in the estimation provided with a number from 1 to 5. After the experiment, the results reported by Lee et al. showed that only slight improvements are detected on the job runtime estimates, despite users making their best effort to perform accurate predictions.

Hirales-Carbajal et al. [9] performed an experimental study of scheduling strategies on grid systems. The authors proposed a scheduling approach considering users runtime estimations and multiple optimization criteria. An offline version of the scheduling problem was solved, i.e., considering all information about tasks and resources is known in advance. The proposed scheduling strategies include the following stages: i) labeling jobs according to users runtime estimates; ii) allocate resources based on optimization criteria; and iii) prioritize jobs. The experimental analysis was carried out using a parametric workload generator and the performance of the proposed scheduler was compared with known single workflow algorithms. The authors considered machine heterogeneity in realistic grid systems, nevertheless a specific model is assumed in order to perform the scheduling evaluations in a repeatable and controllable manner.

Ramírez-Alcaraz et al. [10] showed that user runtime estimates do not help to improve the performance of schedulers in cluster systems. The authors stated that "… an appropriate distribution of job processor requirements over the grid has a higher performance than an allocation of jobs based on user runtime estimates …". Similar conclusions, but related to scheduling to optimize energy consumption, were also found by Iturriaga et al. [15].

In our previous article [4], we introduced the PSP scheduling technique. The main idea of PSP is to integrate user estimates in order to improve the resource utilization and reduce the waiting times of jobs. In PSP, we assign a priority to a job according to the historical accuracy of the runtime estimates of the job owner. In this previous work, user runtime estimates are employed to build a history-based prediction model that is later used by the job scheduler. In this way, user runtime estimates were used as kill times, whereas the predicted time is used to build the schedule. In this previous work, we also assigned a priority from 1 to 5 according to the historical accuracy. The prediction model in PSP uses a history window size. For each newly arrived job, the priority is assigned according to the accuracy of runtime estimates for the owner, which is computed as the average accuracy considering only the last ten completed jobs. The history window approach was originally presented in the article by Tsafrir et al. [11], where authors selected the previous two jobs submitted by the same user. Applying this strategy, the results of Tsafrir et al. showed that taking into account the more recently submitted jobs allows computing improved schedules, as these jobs provide more useful information than older ones.

Regarding the experimental evaluation, in our previous article [4] we created workloads according to the main characteristics of user estimates and jobs submitted to the Cluster FING HPC facility at Facultad de Ingeniería, Universidad de la República [12]. The main results of our previous work showed that using PSP in an environment where some users improve their job runtime estimates over the time is a promising approach to reach the important goal of every user experiencing a decrease on the waiting time of his jobs.

In this article, we improve our previous work by considering a new approach for defining the intervals of accuracy for users estimations, and an aging scheme that increases the priority of jobs over time to prevent starvation.

3 The proposed Penalty Scheduling Policy with aging

The main idea of the penalty policy applied in PSP is based on affecting the priority of jobs according to the historical precision of runtime estimates provided by the users on previous job submissions. The proposal we introduce here extends the method in our previous work [4], for dealing with low accuracy in user jobs estimates.

In our previous work [4], the accuracy of the job runtime estimate by users was defined as $A = \frac{t_{run}}{t_{req}}$, where t_{run} is the real runtime of the job, and t_{req} is the time requested by the user during job submission. The accuracy values are between 0.0 and 1.0, thus, the average accuracy for all submitted jobs is also between these values. The bigger the average accuracy, the better the user estimates the runtime of his jobs, and the PSP scheduling method will assign higher priority to new jobs submitted by the user.

In this article, we propose defining ten levels of accuracy (g1 – g10). The levels of accuracy are defined by the intervals shown in Table 1. The priority is a number between one and forty-nine, where a higher number means that the job is closer to the head of the queue of pending jobs. For example, a job whose owner has an accuracy of 0.17 will have an initial priority of 25. We use these levels of accuracy to assign the initial priority to the submitted jobs. After that, the priority is updated by applying an aging scheme.

PSP calculates the current accuracy of a user by computing the average accuracy for an amount of completed jobs per user. Below, we report results on problem instances in which the accuracy of user at estimating job runtime is calculated for the last ten jobs that were completed.

Table 1. Accuracy intervals of estimates and priorities for each tag name.

Tag name	Accuracy interval	Priority
g1	(0.00,0.05)	1
g2	[0.05,0.10)	10
g3	[0.10,0.15)	20
g4	[0.15,0.20)	25
g5	[0.20,0.30)	30
g6	[0.30,0.40)	35
g7	[0.40,0.52)	40
g8	[0.52,0.64)	43
g9	[0.64,0.78)	46
g10	[0.78,1.00]	49

We use an aging scheme to increase the priority of the jobs over time. The purpose of the aging scheme is to prevent the starvation of jobs. The aging scheme is recursively computed by the expression in Equation 1.

$$\begin{cases} priority_{j,i+1} = g_j + (priority_{j,i} \times \frac{t\_waiting_{j,i}}{t\_estimated_j}) \\ priority_{j,1} = g_j \end{cases} \tag{1}$$

In Equation 1, the new value for the priority of a job is calculated based on the priority of that job in the previous time step. When job j is submitted, the value of the priority in time step i=1 is calculated using PSP. The new value of the priority for task j in time step (i+1) is $priority_{j,i+1}$; g_j is the level of accuracy of task j; and $\frac{t\_waiting_{j,i}}{t\_estimated_j}$ is the normalized waiting time for job *j* in time step *i*. No further historical information is used.

Fig. 1 presents an example of the application of the aging scheme on four different jobs that were submitted at the same time. The new priority for each pending job is calculated at each time step. In this work, we set the time step to 150 seconds, according to the analysis explained in the next paragraph.

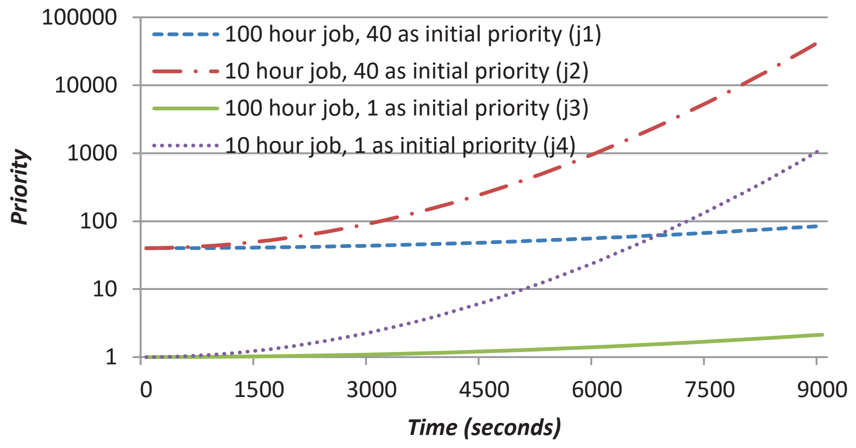

Fig. 1. Example of priority functions for four different types of jobs with a time step of 150 seconds.

The graphic in Fig. 1 shows the increment in the priority values for each job over time. The horizontal axis represents the time in seconds, and the vertical axis represents the priority [in logarithmic scale]. The priority of small jobs increases faster than the priority of large jobs. This happens due to the fact that priority is calculated using the normalized waiting time of jobs. The larger the job (in terms of time requested) the slower the increment of its priority. In Fig. 1 we compared the increment of jobs priority over time. With a time step of 150 seconds job j4 has a priority of 58 after 6750 seconds. Job j1 has a priority of 61 after 6750 seconds. Job j4 waits 18.8% of the requested time before having higher priority than job j1.

In a 1-hour job a 150 seconds time step is 4.2% of its requested time. This length of time step does not impact negatively on user experience. It is a reasonable tradeoff between CPU usage to perform the planning and the length of the jobs in the workloads. Therefore, we decided to use a time step of 150 seconds for the rest of the experimental evaluation.

The proposed PSP+AGING scheduling method was coded and included in the multifactor implementation of SLURM Scheduler Priority Plugin API [14]. The priority API is used by the SLURM job manager, which is the component that accepts jobs requests and includes pending jobs in a priority ordered queue.

The priority calculation process is shown in Figure 2. When a new job arrives, its initial priority is calculated using the PSP+AGING scheduling method. Every time the SLURM priority algorithm is executed, the job priorities are increased according to the aging scheme proposed. When the priority is high, and the resources are available, the job will change its state to running and the job and will leave the queue of pending jobs.

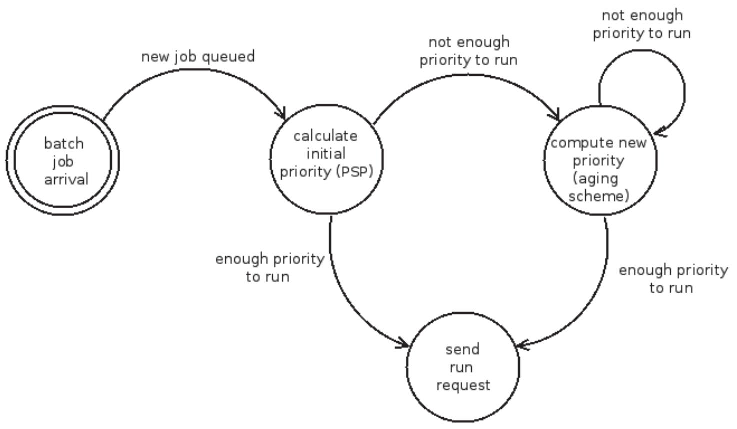
batch job arrival
new job queued
calculate initial priority (PSP)
not enough priority to run
compute new priority (aging scheme)
not enough priority to run
enough priority to run
enough priority to run
send run request

Fig. 2. Diagram of the process for priority calculation in PSP+AGING.

4 Workload analysis and problem instances

We built several problem instances based on real workload traces, to be used in the experimental evaluation of the proposed scheduler. This section summarizes the main findings of the workload characterization study, with special focus on job runtime estimates, and also describes the problem instances generated.

4.1 Workload analysis

We based our analysis in two workload logs taken from the Parallel Workload Archive (PWA) [6]. One of the workload logs was taken from the Curie Supercomputer (Atomic

Energy Commission, France) [from February 2011 to October 2012]. The other workload log was taken from the University of Luxembourg Gaia Cluster [from February 2011 to October 2012]. Both workload logs were provided by Joseph Emeras and are publicly available to download at PWA. In addition, the versions used on our simulations were sanitized according to the procedure described by Feitelson and Tsafrir [13].

The log from CEA Curie Supercomputer has 312,826 jobs and 582 users. A total number of 11,808 Intel processors and 288 NVidia GPU cards are available. According to the information available at PWA, the log from the Gaia Cluster at University of Luxembourg has 51,987 jobs submitted by 84 users and started its operations in 2011. Regarding both workload logs, we built problem instances using a subset of the total number of jobs of each log. The problem instances are specific to each supercomputing infrastructure. In addition, we took into account only jobs that use more than one core.

Table 2 shows the proportion of jobs in the workload logs according to the number of cores requested for the CEA Curie and the Gaia clusters. In this table, it can be seen that 60% of the jobs submitted for the CEA Curie cluster requested between 2 and 16 cores. In the case of the Gaia cluster, 87% of the jobs submitted requested 2 and 16 cores.

Table 2. Cores requested proportions for the CEA Curie and the Gaia clusters.

Cores requested	*Proportion*	
	CEA Curie	Gaia
2 to 16	0.60	0.87
17 to 64	0.07	0.11
65 to 256	0.18	0.02
257 to 1024	0.14	0.00
1025 or more	0.01	0.00

We computed the initial priority of each submitted job applying the model described in Section 3. Table 3 shows the proportions of jobs for each initial priority group. It can be seen that the accuracy for 18% of CEA Curie cluster users is between 20% and 30%. In the case of Gaia cluster, the accuracy for 27% of the users is between 0% and 0.05%. The priority values and the accuracy interval associated to each tag name are shown in Table 1 of Section 3.

Table 3. Job priority groups and its proportions of the CEA Curie and the Gaia clusters.

Tag name	*Proportion*	
	CEA Curie	Gaia
g1	0.14	0.27
g2	0.07	0.08
g3	0.09	0.10
g4	0.10	0.05
g5	0.18	0.15
g6	0.16	0.15
g7	0.11	0.15
g8	0.07	0.00
g9	0.04	0.00
g10	0.04	0.05

4.2 Problem instances

Using the information gathered in the workload analysis, we extracted a set of representative jobs from each workload log in order to make the workload instances. We also created six scenarios to simulate different scheduling policies and user behavior. The set of jobs and the scenarios were chosen to evaluate the proposed scheduler.

The jobs included in the set of jobs are in the order they appear in the workload log and they were chosen so that it was possible to stress the simulated infrastructure. Periods of less intense jobs arriving are also contemplated. The scenarios are characterized by a combination of scheduling policy applied and job runtime estimation considered.

Three scheduling policies are evaluated: (a) FCFS, (b) PSP without job aging and (c) PSP+AGING. Two type of job run time estimations are considered: perfect estimations and real user estimations. Table 4 shows the configurations assigned to each scenario.

Table 4. Scheduling policy and job estimates accuracy configured for each scenario.

Scenario	*Scheduling policy*	*Job estimates*
1	FCFS	
2	PSP without aging	Perfect estimates
3	PSP+AGING	
4	FCFS	
5	PSP without aging	Real user estimates
6	PSP+AGING	

We considered scenarios with perfect estimations in order to have best-case situations for each scheduling policy.

Scenarios 1 and 4 are used as the baseline for the comparison with scenarios 2, 3, 5, and 6.

According to information in the PWA, in the workload logs of CEA Curie and Gaia the utilization of the clusters was 29.3% and 47.9%, respectively. Taking into account the low system utilization, and in order to have a considerable number of jobs in the pending job queue, we configured a simulated infrastructure for each cluster that has a subset of the total number of processors available. It was chosen so that jobs are not necessarily executed as soon as they are submitted.

A total number of 4096 cores comprise the simulated infrastructure for CEA Curie Supercomputer. It was achieved using 1024 CPUs with 4 cores each. For the Gaia Cluster the simulated infrastructure contains 1024 cores. It was achieved using 256 CPUs with 4 cores each. No data about memory capacity of each CPU and memory demand of each job was available when preparing the simulations. Thus, these characteristics are not taken into account in order to run the simulations.

5 Experimental analysis

In this section we report the experimental analysis of the proposed PSP+AGING algorithm. We explain the metrics used to perform the comparison between the six scenarios defined for each workload. Then, we show the characteristics of the computational platform used to run the simulations. We end the section presenting and comparing the numerical results obtained after the simulations.

The goal of the experimental analysis is to evaluate the efficacy of PSP+AGING. We achieve this goal by computing the makespan and the average waiting time for all the scenarios defined in section 4. Then, we compared the results obtained between scenarios, and also between scenarios of different workloads.

All the simulations were performed in a dual-core machine with 8GB RAM, running Ubuntu 14.10. We used the SLURM Simulator to run the simulations for each workload log. SLURM is free software and can be downloaded from [14]. We installed the software in Ubuntu OS as it was recommended by the developers of the tool.

Table 5. Ratio of waiting time between the six scenarios of both workloads. Average waiting time of jobs in the simulation, reported in minutes.

Scenario	*1*	*2*	*3*	*4*	*5*	*6*
Average waiting time (minutes) CEA Curie	4,569	3,785	2,617	4,609	3,706	**2,336**
Average waiting time (minutes) Gaia Cluster	4,358	3,716	2,641	5,153	3,354	**2,421**

Regarding the average waiting time, Table 5 shows the numerical results obtained after performing the simulations with the CEA Curie workload and the Gaia Cluster. It is shown the average waiting time in minutes for each scenario (from scenario 1 to 6). For the CEA Curie workload, in scenario 1 (FCFS with perfect estimates) the average waiting time is 4,569 minutes (i.e., 76.15 hours), whereas in scenario 6 (PSP+AGING with real estimates) the average waiting time is 2,336 minutes (i.e., 38.93 hours). The waiting time in scenario 1 for the CEA Curie workload is 1.956 times higher than in scenario 6. For the Gaia Cluster we computed similar results: the waiting time between scenario 1 (i.e., 4,358 minutes) and scenario 6 (i.e., 2,421 minutes) is 1.800 times higher than in scenario 6.

Fig. 3 shows the Average waiting time for each scheduling policy and each job runtime estimates model for the CEA Curie supercomputer workload.

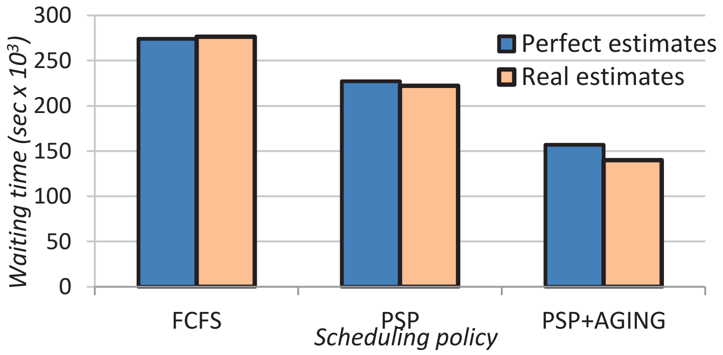

Fig. 3. Average waiting time for each scheduling policy and each job runtime estimates model for the CEA Curie supercomputer workload.

According to the results reported in Fig. 3, users of CEA Curie, in the simulated scenario with a FCFS scheduler and perfect job run time estimates, had a waiting time 1.21 times higher than in the scenario with PSP scheduler and perfect estimates. In the scenario with real estimates, the waiting time in the scenario with FCFS was 1.75 times higher than in the scenario with PSP. In the case of the PSP+AGING and real job run time estimates scenario, the average waiting time is 1.97 times, and 1.57 times lower than in the scenarios with FCFS scheduler and PSP scheduler respectively.

Fig. 4 reports the average waiting times for the Gaia cluster.

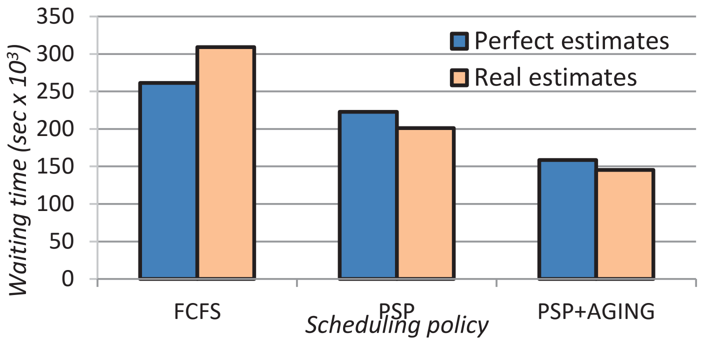

Fig. 4. Average waiting time for each scheduling policy and each job runtime estimates model for the Gaia Cluster workload.

The results reported in Fig. 4 are similar to the ones for the Curie supercomputer. In this sense, the scenario with PSP+AGING and perfect user estimates had an average waiting time that was 1.80 times and 1.54 times lower than the scenarios with PSP scheduler and FCFS scheduler respectively.

Results on both workload logs showed the same trend, in the scenarios with PSP+AGING and real user estimates. Jobs have between 1.5 times and 2.13 times the waiting time in average than scenarios without both PSP and aging.

Table 6. Total makespan for the simulated scenarios.

Scenario	*CEA Curie makespan (days)*	*Gaia makespan (days)*
1	16.19	27.82
2	15.33	27.62
3	17.19	27.79
4	16.17	27.85
5	17.04	28.03
6	17.71	29.08

Table 6 shows the makespan of the each scenario for the subset of jobs selected and simulated of the CEA Curie and the Gaia workload logs. The makespan in the case of the CEA Curie supercomputer varied from 15.33 days in the scenario using FCFS and real user estimates, and 17.71 days in the scenario with PSP+AGING and real user estimates. In the case of the Gaia cluster, the makespan varied between 27.62 and 29.08.

For both workloads used in the simulations, the makespan was higher when using PSP compared to FCFS, and even higher when using PSP+AGING as the scheduling policy. Moreover, the scenario with lower makespan was the one with the simplest scheduling policy and the real (and inaccurate) user estimates By using PSP and aging we achieved lower waiting times. On the other hand, we undertook the makespan of the workloads.

6 Conclusions and future work

In this paper, we present the new scheduling strategy PSP+AGING, that is a variation of the Penalty Scheduling Policy. The new algorithm includes changes in the granularity of accuracy groups and the initial priority of jobs that belong to each group. We also introduced an aging scheme that increments the priority of the jobs in each iteration of the scheduler according to the waiting time normalized with respect to the user estimation of runtime for the job.

We measure the makespan of the workloads to evaluate the benefits of the variation of PSP and the proposed aging scheme for the overall system performance. We also measured the average waiting time of jobs for each workload.

We presented an experimental evaluation in a simulated computer system environment, developed using the SLURM simulator. However, our strategies can be easily included in other popular resource management systems such as Maui.

We analyze the PSP+AGING performance of part of two real workload logs with six scenarios each varying the user runtime estimates and the scheduling policy. The main results show that introducing job priority according to the accuracy of runtime estimation and applying an aging scheme, every user experiences a drop on the waiting time of their jobs. On the other hand, the makespan is not improved.

The lines for future work include extending the experimental analysis with workloads containing a larger quantity of jobs. We also plan on running the simulations with other workloads of the PWA. In this line of work we plan on performing an experimental evaluation on a real environment: the Cluster FING at Facultad de Ingeniería of Universidad de la República.

Regarding the aging scheme we plan on extending the investigation in order to propose new algorithms to compute job aging.

References

1. Cirne, W., Berman, F.: A comprehensive model of the supercomputer workload. IEEE International Workshop on Workload Characterization (2001) 140–148.
2. Tsafrir, D.: Using inaccurate estimates accurately. 15th international conference on Job Scheduling Strategies for Parallel Processing (2010) 208–221.
3. Tsafrir, D., Etsion, Y., Feitelson, D.: Modeling user runtime estimates. 11th international conference on Job Scheduling Strategies for Parallel Processing (2005) 1–35.
4. Rocchetti, N., Iturriaga, S., Nesmachnow, S.: Including accurate user estimates in HPC schedulers: an empirical analysis. XXI Congreso Argentino de Ciencias de la Computación (2015) 1–10.
5. Lee, C. B., Schwartzman, Y., Hardy, J., Snavely, A.: Are user runtime estimates inherently inaccurate? In Workshop on Job Scheduling Strategies for Parallel Processing (2004) 253–263.
6. Feitelson, D.: Parallel Workloads Archive [Online] http://www.cs.huji.ac.il/labs/parallel/workload/ visited 12/07/2016.

7. Ward, W. A. Jr., Mahood, C. L., West, J. E.: Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy. Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing (2002).
8. Su-Hui, C., Arpaci-Dusseau, A., Vernon, M. K.: The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance. Proceedings of the 4th Workshop on Workload Characterization (2002).
9. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., González-García, J. L., Röblitz, T., Ramírez-Alcaraz, J. M.: Multiple workflow scheduling strategies with user runtime estimates on a grid. Journal of Grid Computing (2012) 325–346.
10. Ramírez-Alcaraz, J. M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., González-García, J. L., Hirales-Carbajal, A.: Job allocation strategies with user run time estimates for online scheduling in hierarchical grids. Journal of Grid Computing (2011) 95–116.
11. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Transactions on Parallel and Distributed Systems (2007) 789–803.
12. 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 (1) (2010) 12–15.
13. Feitelson,D., and Tsafrir, D.: Workload sanitation for performance evaluation. In IEEE International Symposium on Performance Analysis of Systems and Software (2006) 221–230.
14. Slurm Simulator web page [Online] https://www.bsc.es/marenostrum-support-services/services/slurm-simulator, visited 12/07/2016.
15. Iturriaga, S., García, S., and Nesmachnow, S. An empirical study of the robustness of energy-aware schedulers for high performance computing systems under uncertainty. In Latin American High Performance Computing Conference, 143-157, 2014.