Improving Backfilling by using Machine Learning to predict Running Times

Eric Gaussier, David Glesser, Valentin Reis, Denis Trystram

Improving Backfilling by using Machine Learning to predict Running Times

Eric Gaussier
University Grenoble-Alpes,
LIG, France
eric.gaussier@imag.fr

David Glesser
BULL – HPC division,
Grenoble, France
david.glesser@bull.net

Valentin Reis
University Grenoble-Alpes,
LIG, France
academic@valentinreis.com

Denis Trystram
University Grenoble-Alpes,
LIG, France
trystram@imag.fr

ABSTRACT

The job management system is the HPC middleware responsible for distributing computing power to applications. While such systems generate an ever increasing amount of data, they are characterized by uncertainties on some parameters like the job running times. The question raised in this work is: *To what extent is it possible/useful to take into account predictions on the job running times for improving the global scheduling?*

We present a comprehensive study for answering this question assuming the popular EASY backfilling policy. More precisely, we rely on some classical methods in machine learning and propose new cost functions well-adapted to the problem. Then, we assess our proposed solutions through intensive simulations using several production logs. Finally, we propose a new scheduling algorithm that outperforms the popular EASY backfilling algorithm by 28% considering the average bounded slowdown objective.

Keywords

High Performance Computing, Running Time Estimation, Scheduling, Machine Learning

1. INTRODUCTION

1.1 Context

Large scale high performance computing (HPC) platforms are becoming increasingly complex. There exists a broad range and variety of HPC architectures and platforms. As a consequence, the *job management system* (which is the middleware responsible for managing and scheduling jobs) should be adapted to deal with this complexity. Determining efficient allocation and scheduling strategies that can deal with complex systems and adapt to their evolutions is a strategic and difficult challenge.

More and more data are produced in such systems by monitoring the platform (CPU usage, I/O traffic, energy consumption, *etc.*), by the job management system (*i.e.*, the characteristics of the *jobs* to be executed and those which have already been executed) and by analytics at the application level (parameters, results and temporary results). All this data is most of the time ignored by the actual systems for scheduling jobs.

The technologies and methods studied in the field of *big data* (including Machine Learning) could and must be used for scheduling jobs in the new HPC platforms.

For instance, and this will be the focus of this paper, the running time of a given job on a specific HPC platform is usually not known in advance and moreover, it depends on the context (characteristics of the other jobs, global load, *etc.*). In practice, most job management systems ask the users for an upper bound on the job running time, threatening to kill it if it exceeds this requested value. This leads to very bad estimates of the running times given by the users [23]. A precise knowledge of the running times is even more important since the algorithms used in most of these systems assume that this value is perfectly known. Thus, it is crucial to determine how to estimate the running times in order to improve scheduling. We believe that there is a huge potential gain in studying this question more deeply and provide more efficient scheduling mechanisms.

Obviously, the job running time is not the only parameter impacted by uncertainties. We focus on it as a proof of concept in order to show that it is possible to improve the scheduling performances for popular FCFS-BF (First Come First Serve with Backfilling) batch scheduling policy. The analysis provided on this work can be extended easily to other scheduling policies.

1.2 Contributions

The main question addressed in this work is to determine to what extent predictions of the running times may help for obtaining a better schedule. For this purpose, we rely on on-line regression methods and consider a family of loss (or cost) functions that are used to learn the prediction model. Then, we show how to use the predictions obtained by improving popular scheduling algorithms. Finally, we perform an experimental evaluation of the proposed new algorithms using several actual log datas on various platforms. The re-

ACM ISBN 978-1-4503-3723-6/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2807591.2807646

sults show an average gain of 28% compared to the classical EASY policy (with a maximum of 86%) and 11% in average compared to EASY++.

1.3 Content

We start by describing the problem in Section 2. Then, we present and discuss in Section 3 the main existing approaches studied so far in parallel processing field for predicting running times and their use in scheduling. Section 4 recalls the main prediction approaches in machine learning and presents an original method for estimate the running time of the jobs based on linear regression. Then, we present the studied scheduling algorithms and their adaptation to predictive techniques in Section 5. Section 6 reports the simulations done with actual log data. They show that some combinations of prediction and scheduling algorithms improve the system performances.

2. PROBLEM DESCRIPTION

2.1 Job management

We are interested in this work in scheduling jobs in HPC platforms. The application developers (or users) submit their jobs in a centralized waiting queue. The job management system aims at determining a suitable allocation for the jobs, which all compete against each other for the available computing resources. In most HPC centers, these users are requested to provide some information about their applications in order to help the system to take better decisions. In particular, it is expected that they give an estimation of the running times. As a job is killed if its actual running time is greater than its requested running time, users tend to significantly increase the duration estimates [23].

Most available open-source and commercial job management systems use an heuristic approach inspired by backfilling algorithms. The job priority is determined according to the arrival times of the jobs. Then, the backfilling mechanism allows a job to run before another job with a highest priority only if it does not delay it. There exist several variants of this algorithm, like conservative backfilling [14] and EASY backfilling [9]. In the former, the job allocation is completely recomputed at each new event (job arrival or job completion) while in the second, the process is purely on-line avoiding costly recomputations.

SLURM is the most popular job management system [25]. In the last release of the TOP500 ranking in November 2014, SLURM was performing job management in six of the ten most powerful systems, including the number one. It uses EASY backfilling and includes the possibility to sort the waiting jobs according to various priorities (like by increasing age, size or share factors, *etc.*). Other job management systems including MOAB [12], TORQUE [22], PBSpro [17] or LSF [11] are built upon the same approach with their own specificities. A review on job schedulers can be found in Georgiou [7].

We focus on EASY backfilling as a basic mechanism for assessing our approach. In the remaining of the paper, the corresponding policy will be denoted in short by EASY.

2.2 Dealing with uncertainties

The objective of this section is to show by some simulations on actual data that using good running time predictions significantly improves the scheduling performances. First, let us clarify the vocabulary: a job is *over-predicted* if its predicted running time is greater than the actual running time. Similarly, a job is under-predicted when the predicted running time is lower than the actual running time.

Table 1: AVEbsld performances of EASY **(using requested times) and** EASY-Clairvoyant **(using actual running times). Values between parentheses show the corresponding decrease in AVEbsld.**

Log	EASY	EASY-Clairvoyant
KTH-SP2	92.6	71.7 (22%)
CTC-SP2	49.6	37.2 (25%)
SDSC-SP2	87.9	70.5 (19%)
SDSC-BLUE	36.5	30.6 (16%)
Curie	202.1	69.9 (65%)
Metacentrum	97.6	81.7 (16%)

Table 1 reports the comparison of simulations based on the testbed detailed in Section 6. Each log runs with EASY, first with the original user's requested running times, then with the actual running times (as if the users were entirely clairvoyant). The metric used for comparison, AVEbsld is described in Subsection 5.3.

The results reported in Table 1 emphasize that clairvoyant simulations are in average 27% better than simulations using the original requested running times. As running times are shorter in the clairvoyant case, more jobs can be backfilled and thus, the performances are improved. Taking into account actual running time values always improves the scheduling performances, thus accurate running time estimates are crucial for reaching good performances.

2.3 Formulation of the problem

The problem studied in this work is to execute a set of concurrent parallel jobs with rigid resource requirements (it means that the number of resources required by a job is known in advance) on a HPC platform represented by a pool of m identical resources (we do not assume any particular interconnection topology). The jobs are submitted over time (on-line).

There are n independent jobs (indexed by integers), where job j has the following characteristics:

- Submission date r_j (sometimes called *release date*);
- Resource requirement q_j (processor count);
- Actual running time p_j (sometimes called *processing time*);
- Requested running time $\widetilde{p_j}$, which is an upper bound on p_j;
- Additional data that has no direct impact on the physical description of the job (*e.g.* the time of the day when the job is submitted or the executable name). This data may be used to learn about jobs, users and the system.

The resource requirement q_j of a job is known when the job is submitted at time r_j, while the running time p_j is given

as an estimate. Its actual value is only known *a posteriori* when the job really completes.

The problem is to design several algorithms that predict the running times in order to provide good scheduling performances.

3. RELATED WORK

3.1 Prediction

Historically, job running time prediction has been first attempted [8] by categorizing the jobs according to a predefined set of rules. Then, statistics based on such job's category are used to generate a prediction. In this approach, called *Templates*, a partitioning into templates has to be provided by the job management system or the system administrator. It can be seen as an ancestor of tree-based regression models in which the binning has to be obtained trough statistical analysis of the specific system and population and/or discussion with a domain expert. The technique was subsequently adapted [21] using a more automatic way (a genetic algorithm evolving template attributes) to generate the rules. These works used minimal, high-level information about jobs similar to what can be found in HPC logs.

There exist other works that use more specialized methods, but require the modeling of the jobs. For instance, Schopf *et al.* predict running time of applications by analyzing their functional structure [20]. An another example is the method developed by Mendes *et al.* which performs static analysis of the applications [13].

A later survey [10] evaluates the use of more recent supervised learning tools. This work focuses on two scientific applications and uses in-depth information about both the jobs (*e.g.* input parameters) and the machines (*e.g.* disk speed). A closely related paper by Duan *et al.* [3] proposes an hybrid Bayesian-neural network approach to dynamically model and predict the running time of scientific applications. It uses in-depth information about jobs and their environment as well.

All these previous approaches assume jobs and their running times to be identically distributed and independent (*i.i.d.*), and therefore, they do not leverage dependencies between job submissions. A stochastic model [15] has been proposed for predicting job running time distributions. By opposition to the previous studies which only used job descriptions, this technique only relies on historical running time information. It treats successive running times of a given user as the observations of a Hidden Markov Model [18], and hence it does not use the hypothesis that job submissions would be *i.i.d.*.

3.2 Scheduling based on Predictions

There are only a few methods for performing scheduling using predictive techniques.

The Hidden Markov Model used in [15] is a probabilistic generative model, and as such it is possible to easily obtain the conditional distribution of the running time of a job. As a consequence, it is possible to design scheduling algorithms that use job running time distributions as an input [16].

The most relevant work for scheduling jobs on large parallel systems using predictions is [24], in which the average of the two last available running times of the job's user is used as a prediction. It leads to surprisingly good results given its simplicity. This work also introduces an improved version of EASY: EASY++. This algorithm use the predicted value for the backfilling. The waiting jobs are considered by their order of arrival, but during the backfilling phase jobs are sorted shortest first. They also introduce a correction mechanism: when the the prediction technique under-estimates a running time, a new estimation of the running time has to be obtained. The proposed correction mechanism is simple: the authors add a fixed amount of time from a predefined list of values each time they under-estimate.

To the best of our knowledge, there is no other work that relies on state-of-the-art machine learning methodology for running time prediction and evaluates the resulting scheduling. In the following two sections, we present prediction and scheduling mechanisms.

4. PREDICTION METHOD

We first outline in this section the characteristics that the prediction method should have, prior to describing our approach in detail.

4.1 Rationale

Let us first argue that an approach based on machine learning for predicting the running times of jobs should have the following characteristics:

It should be based on minimal information. In hope that results extend well to new HPC systems and be usable in mainstream job management system, a learning-based system should prove its effectiveness on minimal job descriptions. In this light, one reasonable approach is to use information of the Standard Workload Format (*SWF*) [5].

It should work on-line. This is motivated by previous studies which emphasize that dependencies between job running times are so far from being *i.i.d.* that *two successive running times* [24] are enough to predict running time with good accuracy. Therefore, an algorithm based on batches (which re-approximates the learned concept once every k jobs, where $k \gg 1$) should not be favored.

It should use both job description and system history Unlike previous studies that either assume jobs to be temporally independent or rely exclusively on running time locality [15], a holistic approach should leverage both job descriptions and temporal dependencies.

It should be robust to noise. Because HPC jobs can have an erratic behavior (*e.g.* they may fail or hang-up), the data one is relying on will be noisy. The learning algorithm should be robust to this noise and avoid overfitting.

It should accept an arbitrary loss function. Attempting to minimize the cumulative loss of an arbitrary function allows for a "declarative" statement of the harm incurred by a misprediction. This last aspect is motivated by the intuition that an inaccurate prediction of a job's running time would not harm the scheduling in an identical way depending on the job's characteristics and the direction of the error. This will be explored

in detail in the next Subsection. Arbitrary loss functions generally pose computational limitations, and in practice convex loss functions are often used.

We now turn to the description of the prediction method.

4.2 A new prediction approach

A job j is represented by a vector[1] $\mathbf{x}_j \in \mathbb{R}^n$ where n is the number of features of the model. The features which we feed the algorithm with are described in Table 2. These features are taken from various sources of information, such as the job's description ($\widetilde{p_j}$ and q_j). Others are taken from historical information (*e.g.* $p_{j-1}^{(k)}$) and some are taken from the current state of the system (*e.g.* Jobs Currently Running). Additionally, some features are taken from the environment (*e.g.* Time of the day).

The prediction is achieved via a ℓ_2-regularized polynomial model. This choice is motivated by the availability of highly robust algorithms to fit on-line linear regression models [19], even in the presence of an adversary scaling of the features. The ℓ_2 norm regularizer is used to prevent the quadratic model from overfitting and the polynomial representation (here of degree 2) to take into account dependencies between features. The regression function is:

$$f(\mathbf{w}, \mathbf{x}) = \mathbf{w}^\intercal \Phi(\mathbf{x}) \quad \mathbf{w} \in \mathbb{R}^{1+2n+\binom{n}{2}} \tag{1}$$

where the w_i are the parameters (to be learned) of the model, and Φ is a vector of basis functions:

$$\Phi(\mathbf{x}) = (1, x_1, \cdots, x_n, x_1x_2, \cdots, x_kx_l, \cdots, x_{n-1}x_n)^\intercal$$

Denoting the actual running time of job j by p_j, the cumulative loss for up to the N-th already-processed job is[2]:

$$\sum_{j=1}^{N} \mathfrak{L}(\mathbf{x}_j, f(\mathbf{x}_j), p_j)$$

where $\mathfrak{L}(\mathbf{x}, f(\mathbf{x}), y)$ is the loss function associated with predicting a value of $f(\mathbf{x})$ in the case where the actual running time is y. The regression problem finally takes the form:

$$\arg\min_{\mathbf{w}} \sum_{j=1}^{N} \mathfrak{L}(\mathbf{x}_j, \mathbf{w}^\intercal \Phi(\mathbf{x}_j), p_j) + \lambda ||\mathbf{w}||_2 \tag{2}$$

where λ is the regularization parameter. Once $\mathbf{w}$ has been learned, new running times are predicted through Equation (1).

The choice of the loss function is crucial and not straightforward here as the impact of a bad prediction on the running time varies from job to job as well as from the direction of the error (over- or under-prediction). Indeed, scheduling algorithms behave differently with respect to under-prediction and over-prediction: in the case of an under-prediction, a destructive effect on the planned schedule can happen, while an over-prediction never makes a planned schedule feasible but may imply unused resources. This suggests that one should rely on asymmetrical losses, that can be based on standard loss functions dedicated to either under- or over-prediction. Another source of complication with respect to prediction arises not just from the backfilling strategy, but from the scheduling problem itself. The tasks have a two-dimensional representation based on (q_j, p_j) and it is reasonable to expect that the difficulty of finding a good schedule should be dependent not only on the prediction error, but also on how it is distributed among jobs of different q and p. This suggests that the loss function should be weighted differently according to the jobs considered, leading to:

$$\mathfrak{L}(\mathbf{x}_j, f(\mathbf{x}_j), p_j) = \begin{cases} \gamma_j . L_u(f(\mathbf{x}_j) - p_j) & \text{if } f(\mathbf{x}_j) \geq p_j \\ \gamma_j . L_o(p_j - f(\mathbf{x}_j)) & \text{if } f(\mathbf{x}_j) < p_j \end{cases}$$

where L_u is the underprediction basis loss function, L_o is the overprediction basis loss function, and $\gamma_j > 0$ corresponds to the weight of job j.

Figure 1 shows an example of an asymmetrical loss function with a unit weight γ_j, using a squared loss basis for underprediction and a linear loss basis for overprediction.

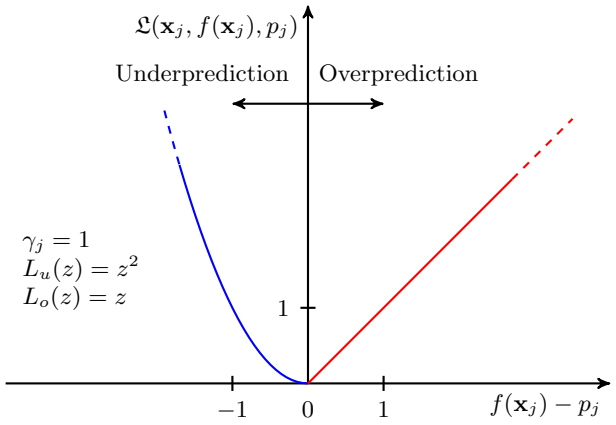

Figure 1: Example Loss function $\mathfrak{L}$, plotted with respect to the difference of it's second and third parameters $f(\mathbf{x}_j) - p_j$ (the prediction error).

In this study, we consider two standard loss functions for under- and over-prediction, namely the squared loss ($L(z) = z^2$) and the linear loss ($L(z) = z$). It can be verified that all the possible combinations of these two loss functions in $\mathfrak{L}$ are continuous and convex (even though not differentiable everywhere) with respect to vector $\mathbf{w}$.

Choosing the weighting factor γ_j is not straightforward. On the one hand, a key property of backfilling algorithms is that jobs of small area (small p, small q) are easier to backfill. Underpredicting small jobs can therefore mean delaying a reservation, and one should therefore use a weighting factor which *decreases* with p and q. On the other hand, as we consider systems with no preemption, once a large (large p, large $q \approx m$) job is started, almost no resources remain available. Thus, jobs in the queue are doomed to wait a long time. It follows that predicting jobs of large area correctly should be beneficial, and one should therefore use a weighting factor which *increases* with p and q. To account for these various elements, we explore four different possibilities along with a constant weighting factor, all of which are shown in Table 3.

Finally, for each choice of two basis loss function L_u and

[1] As common in machine learning, we use bold letters to denote vectors and standard letters for scalars.

[2] Note that one can also consider the k latest jobs or weigh differently the jobs to favor recent ones. These variants are straightforward to consider from the framework developed here.

Table 2: Features extracted from the *SWF* data, for job j, belonging to user k.

Feature	Meaning
$\widetilde{p_j}$	the time the user requested for her job.
$p_{j-1}^{(k)}$	the running time of the last job of the same user, or 0 if such a job does not exist.
$p_{j-2}^{(k)}$	the running time of the second-to-last job of the same user, or 0 if N/A.
$p_{j-3}^{(k)}$	the running time of the third-to-last job of the same user, or 0 if N/A.
$AVE_2^{(k)}(p)$	the average running time of the two last historically recorded jobs of the same user.
$AVE_3^{(k)}(p)$	the average running time of the three last historically recorded jobs of the same user.
$AVE_{all}^{(k)}(p)$	the average running time of all historically recorded jobs of the same user.
q_j	amount of (CPU) resource requested by job j.
$AVE_{hist,r_j}^{(k)}(q)$	average historical resource request of user k, taken at release date of job j.
$\frac{q_j}{AVE_{hist,r_j}^{(k)}(q)}$	amount of resource requested normalized by average resource request.
$AVE_{curr,r_j}^{(k)}(q)$	average resource request of the user's currently running jobs, at release date
Jobs Currently Running	number of jobs of the user running, at release date
Longest Current running time	longest running time (so-far) of the user's currently running jobs, at release date
Sum Current running times	sum of the running times (so-far) of the user's currently running jobs, at release date
Occupied Resources	total size of resources currently being allocated to the same user.
Break Time	time elapsed since last job completion from the same user.
$\begin{cases} cos(\frac{2*\pi}{t_{day}} * (r_j \mod t_{day})) \\ sin(\frac{2*\pi}{t_{day}} * (r_j \mod t_{day})) \end{cases}$	time of the day the job was released. The periodic feature is decomposed into its cosinus and sinus, using the day period t_{day} (length of a day in seconds)
$\begin{cases} cos(\frac{2*\pi}{t_{week}} * (r_j \mod t_{week})) \\ sin(\frac{2*\pi}{t_{week}} * (r_j \mod t_{week})) \end{cases}$	time of the week the job was released. The periodic feature is decomposed into its cosinus and sinus, using the week period t_{week} (length of a day in seconds)

Table 3: Weighting factors considered for training the model. The constants are chosen to ensure positivity of the weights with typical running times and resource requests in the HPC domain. Logarithms are used to alleviate the high range produced by ratios.

γ_j	Interpretation
1	Constant weight.
$5 + \log(\frac{q_j}{p_j})$	Short jobs with large resource request should be well-predicted.
$5 + \log(\frac{p_j}{q_j})$	Long jobs with small resource request should be well-predicted.
$11 + \log(\frac{1}{q_j.p_j})$	Jobs of small "area" should be well-predicted.
$\log(q_j.p_j)$	Jobs of large "area" should be well-predicted.

L_o along with weighting scheme γ_j, the regression model (*i.e.* the vector $\mathbf{w}$) is learned on an on-line training/testing set by minimizing the cumulative loss through the Normalized Adaptive Gradient [19] algorithm (NAG), a variant of the classical Stochastic Gradient Descent [1]. The NAG algorithm poses the advantage of being robust to adversarial[3] scaling of the features. Robustness to feature scaling is a requirement of our problem because some features are difficult or impossible to normalize (*e.g.* Break Time is unbounded). Section 6 describes the data sets retained as well as the values of the parameters considered for the search. Note that when $\gamma_j = 1$ and $L_u(z) = L_o(z) = z^2$, one is just considering a standard squared loss regression problem, learned in an on-line manner.

5. SCHEDULING

5.1 The EASY algorithm

[3]The notion of *adversary* simply means that the scaling can be arbitrary. See [2] for a comprehensive study.

As mentioned in Section 2.1, we target *EASY* because of it is broadly used and it has well-established performances.

EASY is one of the most on-line version of backfilling algorithms. Backfilling algorithms are based on a priority queue where the jobs with the highest priorities are launched first. A job with a lower priority can run before a higher priority job if and only if it does not delay it. EASY is considered as an on-line algorithm since the decision to launch a job is only taken at the last moment. For instance as it is depicted in Figure 2, the decision to launch job 2 is taken after the completion of job 1. Job 3 has been backfilled. In this figure, we can measure the importance of running times in backfilling algorithms. If the estimated running time of job 1 was much shorter, job 3 would not be backfilled.

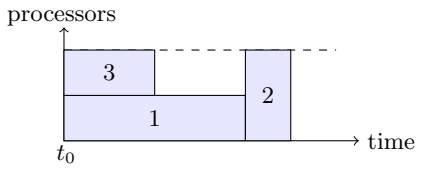

Figure 2: Example of EASY for a queue of 3 jobs ordered according to their index. The decision to launch job 1 is taken at t_0. The second ready job (2) can not be executed since there are not enough resources left. Thus, job 3 can also be launched at t_0. The decision to launch job 2 is finally taken when job 1 and 3 complete.

Tsafrir *et al.* proposed in [24] a slightly modified version of EASY, called EASY-SJBF, which performed better with running time predictions than the standard version. During the phase when the algorithm determines the candidate jobs to be backfilled, the jobs are sorted by increasing predicted running times instead of considering the FCFS order. They argue that this way, more jobs will be backfilled and thus, the overall performances will increase.

5.2 Correction mechanism

In this section, we are interested in the following question: *what happens to the schedule when the running times are mispredicted?*

The case of over-predicted running times is easy to handle by backfilling since the situation is the same as without learning where the users over-estimate the requested running times. In case of under-predicted running times, there are two points to consider. First, we should determine a new prediction for the remaining execution time and second, we have to check whether the correction does not disturb too much the scheduling algorithm. Both points are detailed as follows.

First point. We prefer to update the running times by some simple rules instead of computing again a prediction by the learning scheme, which gave a wrong value. Obviously, these updated running times remain bounded by the requested running times. These values are given by a correction algorithm. We consider the three following ones:

- Requested Time – Set the new prediction value to be $\widehat{p_j}$ (the user requested running time);
- Incremental – Use the corrective technique from [24], *i.e.* increase at each correction by an fixed amount of time (1min, 5min, 15min, 30min, 1h, 2h, 5h, 10h, 20h, 50h, 100h);
- Recursive Doubling – Increase the prediction value by the double of the elapsed running time.

Second point. Do backfilling variants handle the updated prediction? As the considered backfilling algorithms are on-line in nature, they adapt dynamically to the changes. However, notice that under-prediction with backfilling can lead to starvation. For instance, a large job will indefinitely wait for its required resources if under-predicted shorter jobs are systematically backfilled before. They will be launched before the large one, leading to an unbounded delay.

5.3 Objective Functions

As it is commonly admitted [4], the performances of scheduling algorithms are measured using the *bounded slowdown*, which is defined as follows for job j:

$$\text{bsld} = \max\Big(\frac{wait_j + p_j}{\max(p_j, \tau)}, 1\Big)$$

where $wait_j$ is the waiting time of job j (from the time it is released in the system and the time it starts its execution) and τ is a constant preventing small jobs to reach too large slowdown values. In the literature, τ is generally set to 10 seconds. We will use this value in the experiments.

One related objective function usually used for comparing performances is the average of bsld, defined as:

$$\text{AVEbsld} = \frac{1}{n}\sum_j \max\Big(\frac{wait_j + p_j}{\max(p_j, \tau)}, 1\Big)$$

In this paper, all scheduling performances are measured with this objective function.

6. EXPERIMENTS

6.1 Experiment objectives

The simulations we conducted aim at answering the following two questions:

1. Do the proposed predictive and corrective techniques improve existing scheduling algorithms?
2. Which prediction loss function, correction mechanism and backfilling variant work well together?

Previous studies have mainly focused on predicting running times, independently of the scheduling algorithms, using standard measures for the prediction error. In contrast, we aim here at predicting running times *for scheduling jobs with backfilling*, through a combination of appropriate loss functions, correction mechanisms and backfilling variants. The solutions we develop are thus closer to the problem of improving HPC systems. Moreover, identifying efficient combinations of prediction technique, correction mechanism and backfilling variants should provide insights into the behavior of backfilling algorithms when running times are unsure. In the rest of the paper, we refer to such combinations as *heuristic triples*.

Table 4: Workload logs used in the simulations.

Name	Year	# CPUs	# Jobs	Duration
KTH-SP2	1996	100	28k	11 Months
CTC-SP2	1996	338	77k	11 Months
SDSC-SP2	2000	128	59k	24 Months
SDSC-BLUE	2003	1,152	243k	32 Months
Curie	2012	80,640	312k	3 Months
Metacentrum	2013	3,356	495k	6 Months

Table 5: Considered parameter values of the loss function. There are three effective parameters, for a total of 20 combinations.

Parameter	Possible Values
L_u	$z \mapsto z^2$, $z \mapsto z$
L_o	$z \mapsto z^2$, $z \mapsto z$
γ_j	See Table 3 (5 values)

6.2 Description of the testbed

We make use in our study of a set of actual workload logs, described in Table 4. All these workload logs but Metacentrum are extracted from the Parallel Workload Archive [5]. Metacentrum is extracted from the personal website of Dalibor Klusáček[4]. They come from various HPC centers, correspond to highly different environments and have been selected for their high resource utilization, which challenges scheduling algorithms [6]. For each log, we run scheduling simulations using every possible heuristic triples: prediction technique, correction mechanism and backfilling variant.

All simulations are run using a fork of the open-source[5] batch scheduler simulator *pyss*. The source of this forked version is available on-line[6].

All prediction techniques based on the different loss functions and weighting schemes introduced in Section 4 are considered here, in conjunction, for comparison purposes, with the actual running time p_j, denoted as Clairvoyant, the user requested time $\widetilde{p_j}$, denoted as Requested Time and the average of the two previous running times for the jobs of user k, denoted as $AVE_2^{(k)}(p)$. For correction, we rely on the three correction techniques presented in Subsection 5.2: Requested Time, Incremental and Recursive Doubling. Lastly, we rely on the two backfilling algorithms presented in Subsection 5.1 , namely EASY and EASY-SJBF.

Notice that the case where Requested Time is used as prediction technique and EASY as the backfilling variant corresponds to the standard EASY backfilling algorithm. Similarly, the case where Incremental correction method is used with the $AVE_2^{(k)}(p)$ prediction technique and the EASY-SJBF backfilling variant correspond to the EASY++ algorithm introduced by Tsafrir *et al.* [24].

As it is reasonable to expect that scheduling performances due to a loss function (and therefore, a learned model) are dependent on both the backfilling variant and correction mechanism, we evaluate all of them together. This induces a higher complexity and a high number of simulations. For each workload log, the experimental campaign runs 128 simulations. As it is impossible to present all of them in detail, we invite the reader to look at our repository[6].

The experiment campaign contains significantly more heuristic triples than workload logs, and this raises a multiple hypothesis testing problem. Therefore, one should approach the analysis of the results using sound statistical practices.

Subsection 6.3 outlines the best heuristic triple. Afterwards, Subsection 6.4 contains an analysis of the predictions made.

[4]http://www.fi.muni.cz/∼xklusac/

[5]pyss - the Python Scheduler Simulator, available at http://code.google.com/p/pyss/

[6] http://github.com/freuk/predictsim

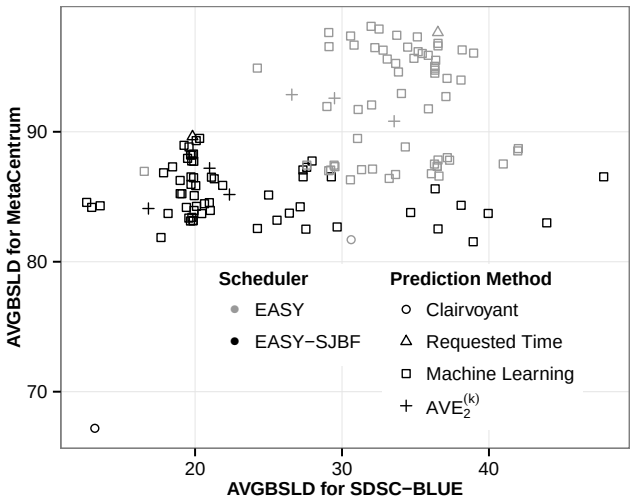

Figure 3: Scatter plot of heuristic's relative performance between the MetaCentrum and SDSC-BLUE logs.

6.3 Which heuristic triple is prevalent?

6.3.1 Raw results

As shown in Table 6 which displays the AVEbsld scores for the different approaches, the results seem promising as they tend to favor approaches based on learning (the Clairvoyant results are reported for comparison purposes only and correspond to an upper bound of what one can expect). However, one should not conclude too hastily, as even though the best approach is always obtained using a predictive-corrective approach (corresponding to the columns EASY with Learning Techniques in Table 6), it is not clear which heuristic triple is prevalent *a priori*.

In particular we observe that the SJBF variant introduced by [24] is rather efficient at leveraging accurate values of the running times, as the Clairvoyant EASY-SJBF algorithm almost always outperforms its competitors.

6.3.2 Correlation between logs

A key part of the analysis of predictive techniques is to see how performances correlate between different systems.

Figure 3 shows the AVEbsld between the MetaCentrum and SDSC-BLUE logs, meant to illustrate the irregularity of performances between logs. We can observe that Clairvoyant EASY-SJBF is the best in both logs, but there is no clear correlation between all the heuristic triples.

The correlation coefficient is measured here with the Pear-

Table 6: Overview of the AVEbsld for each simulations. For predictive techniques, only the best and the worst AVEbsld are given. The best non-clairvoyant heuristic triples are outlined in bold.

	Clairvoyant EASY				EASY with Learning Techniques	
Trace	FCFS	SJBF	EASY	EASY++	FCFS	SJBF
KTH-SP2	71.7	49.8	92.6	63.5	62.6 - 93.2	**51.4** - 74.5
CTC-SP2	37.2	17.6	49.6	85.8	25.5 - 163.5	**16.3** - 134.7
SDSC-SP2	70.5	56.8	87.9	79.4	70.9 - 102.3	**69.7** - 194.8
SDSC-BLUE	30.6	13.2	36.5	21.0	16.5 - 48.0	**12.6** - 47.8
Curie	69.9	12.1	202.1	193.5	26.3 - 9348.8	**24.3** - 4010
Metacentrum	81.7	67.2	97.6	87.2	86.3 - 98.1	**81.5** - 89.8

Table 7: AVEbsld performance of the heuristic triples resulting from cross validation. Values in parenthesis show the AVEbsld reduction obtained respective to EASY**.**

Log	C-V Heuristic triple	EASY	EASY++
KTH-SP2	**51.4** (44%)	92.6	63.5 (31%)
CTC-SP2	**20.5** (59%)	49.6	85.8 (-72%)
SDSC-SP2	**75.0** (15%)	87.9	79.4 (10%)
SDSC-BLUE	34.7 (05%)	36.5	**21.0** (42%)
Curie	**27.9** (86%)	202.1	193.5 (04%)
Metacentrum	**84.2** (14%)	97.6	87.2 (11%)

son's Correlation coefficient, which is computed for each couple of logs. With a mean of 0.26 (min: 0.01, max: 0.80), this coefficient is low. This means that it is not possible to know from the result on one log if a heuristic triple will perform well on another logs. However, one can still try and learn an appropriate heuristic triple from existing systems, as described below.

6.3.3 Triple selection

We consider here a *leave-one-out cross validation* process in which 5 logs are (alternatively) used to select the best triple, the performance of which is evaluated on the 6th log. The idea is to assess whether one can select a good heuristic triple from existing logs. The experiment is repeated six times (for the six logs) and the results are averaged over the six repetitions. The best heuristic triple is the one that optimizes the sum of the AVEbsld on the 5 logs. Table 7 displays the obtained results. As one can note, the results obtained with this selection process, denoted C-V (for cross-validation) heuristic triple, significantly outperforms the EASY and EASY++ approaches on all workloads except SDSC-BLUE.

Even more interestingly, it turns out that the best heuristic triple on all logs using the selection method above is the same[7] and corresponds to the following setting:

Prediction Technique : Regression function described in Section 4 with the loss function:

$$\mathfrak{L}(\mathbf{x}_j, f(\mathbf{x}_j), p_j) = \begin{cases} \log(r_j.p_j).(f(\mathbf{x}_j) - p_j)^2 & \text{if } f(\mathbf{x}_j) \geq p_j \\ \log(r_j.p_j).(p_j - f(\mathbf{x}_j)) & \text{if } f(\mathbf{x}_j) < p_j \end{cases} \quad (3)$$

Correction mechanism : INCREMENTAL

backfilling variant : EASY-SJBF

6.3.4 Summary

We have shown here that one can learn an appropriate heuristic triple from existing logs. This heuristic triple yields better scheduling performances than EASY and EASY++. Furthermore, on the workloads considered, a heuristic triple singles out as it is the one always selected. This heuristic triple obtains an average AVEbsld reduction of 28% compared to EASY and 11% compared to EASY++, and can reduce the AVEbsld by 86% compared to EASY (on the Curie workload for example). This triple uses the INCREMENTAL correction technique and the SJBF queue ordering from [24], as well as a machine learning-based approach with custom loss functions (3). We call this loss function E-Loss (for EASY-Loss) and briefly discuss its behavior in the next Subsection.

6.4 Prediction analysis

The first question one usually asks after having used a predictive technique is, *what is the prediction accuracy*? Experimental results outline that while prediction performance is important, choosing the right loss for the prediction is even more critical. This is observed in Table 8 which shows the prediction errors of both the $AVE_2^{(k)}(p)$ prediction technique and our E-Loss based approach.

Table 8: MAE and E-Loss for different prediction techniques. All values are in seconds.

Prediction Technique	MAE	Mean E-Loss
$AVE_2^{(k)}(p)$	5217	10.2×10^8
E-Loss Learning	6762	2.35×10^5

One can see from these values that while the $AVE_2^{(k)}(p)$ performs well with respect to the Mean Average Error (MAE), its performance on the E-Loss is quite poor.

Equation (3) shows that the E-Loss is an asymmetrical loss function, with a linear branch for under-prediction and a squared branch for over-prediction. Therefore this loss

[7] with one exception: the C-V Heuristic selected for SDSC-SP2 uses the REQUESTED TIME correction mechanism. This could account for it's degraded performance.

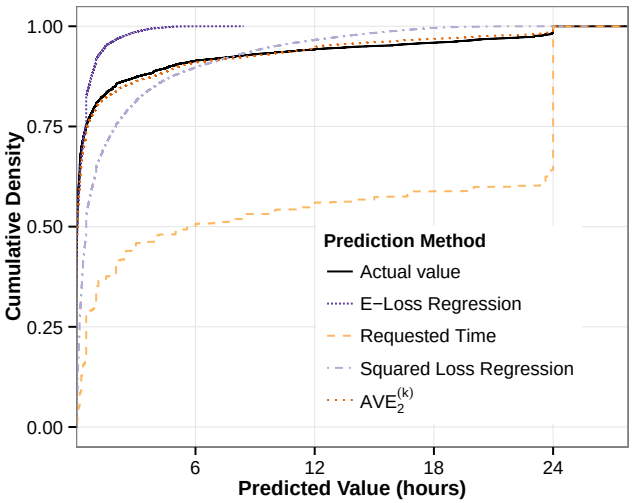

Figure 5: Experimental cumulative distribution functions of predicted values obtained using the Curie log.

function discourages over-prediction. Additionally, the E-Loss uses a weighting factor that increases with the size of jobs in terms of p and q. A helpful visualization for understanding how the E-loss behaves in practice is the empirical cumulative distribution function (ECDF) of the prediction errors produced by the resulting machine learning model. Figure 4 shows the ECDFs of such prediction errors for main prediction techniques. From this figure, one can see the behavior of the E-loss with respect to that of the standard squared loss. The E-loss ECDF is shifted to the left, which means that more under-prediction errors are indeed made than with standard regression. This is coherent with intuition gleaned from the analysis form of the loss function.

Finally, Figure 5 shows the ECDF of the values that were predicted. On this graph, we see that in order to generalize well with respect to the E-Loss, the learning model ends up being strongly biased towards small predictions. This displacement suggests that there might be a beneficial effect to backfilling jobs very aggressively when using EASY-SJBF.

6.5 Discussion

As mentioned above, the proposed approach outperforms EASY++ with a **11%** of reduction in average AVEbsld, and has a reduction of **28%** in average AVEbsld when compared to EASY. This result is obtained by changing the prediction technique of EASY++ to one that uses a custom loss function that we refer to as E-loss. We have furthermore observed that on each log, roughly 0.1% of jobs have extremely high values of bounded slowdowns. Such a behavior is obtained with every heuristic triple based on $AVE_2^{(k)}(p)$ or MACHINE LEARNING prediction techniques. Extreme values seem to be a shortcoming of incorporating predictions without a mechanism for dealing with extreme prediction failures. Moreover, because such failures are often due to jobs that do not run properly, we are confronted here with an evaluation problem, as the cost of such events could be incurred on the user rather than the system. New performance evaluation measures are needed to deal with such problems, especially for schedulers without no-starvation guarantees (as other authors already suggested [6]).

7. CONCLUSION

The purpose of this work was to investigate whether the use of learning techniques on the job running times is worth for improving the scheduling algorithms. We proposed a new cost function for prediction and run simulations based on actual workload logs for the most popular variants of backfilling. The results clearly show that this approach is very useful, as they reduce the average bounded slowdown by a factor of 28% compared to EASY. Moreover, the proposed approach may be extended easily to other scheduling policies.

There are two derived interesting questions that could be studied in the future: First, to extend this study to other learning features. For instance, using a more precise knowledge of the jobs should lead to even better results. Reaching this goal needs either to adapt the simulations or to test on a real system. Second, we would like to go one step further by learning directly on the priority of waiting jobs (using *Learning To Ranks* algorithms) instead of learning on the job features like what we did for the running times.

Acknowledgments.

The authors would like to warmly thank Yiannis Georgiou for his unfailing support. They are also indebted to Krzysztof Rzadca for helpful discussions and support. We thank gracefully the contributors of the Parallel Workloads Archive, Victor Hazlewood (SDSC SP2), Travis Earheart and Nancy Wilkins-Diehr (SDSC Blue), Lars Malinowsky (KTH SP2), Dan Dwyer and Steve Hotovy (CTC SP2), Joseph Emeras (CEA Curie), and of course Dror Feitelson. The Metacentrum workload log was graciously provided by the Czech National Grid Infrastructure MetaCentrum.

The work is partially supported by the ANR project MOEBUS.

8. REFERENCES

[1] L. Bottou. Stochastic learning. In *Advanced lectures on machine learning*. 2004.

[2] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[3] R. Duan, F. Nadeem, J. Wang, Y. Zhang, R. Prodan, and T. Fahringer. A hybrid intelligent method for performance modeling and prediction of workflow activities in grids. In *Cluster Computing and the Grid*, 2009.

[4] D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In *Job Scheduling Strategies for Parallel Processing*. 2001.

[5] D. G. Feitelson, D. Tsafrir, and D. Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 2014.

[6] E. Frachtenberg and D. G. Feitelson. Pitfalls in parallel job scheduling evaluation. In *Job Scheduling Strategies for Parallel Processing*, 2005.

[7] Y. Georgiou. *Resource and Job Management in High Performance Computing*. PhD thesis, Joseph Fourier University, 2010.

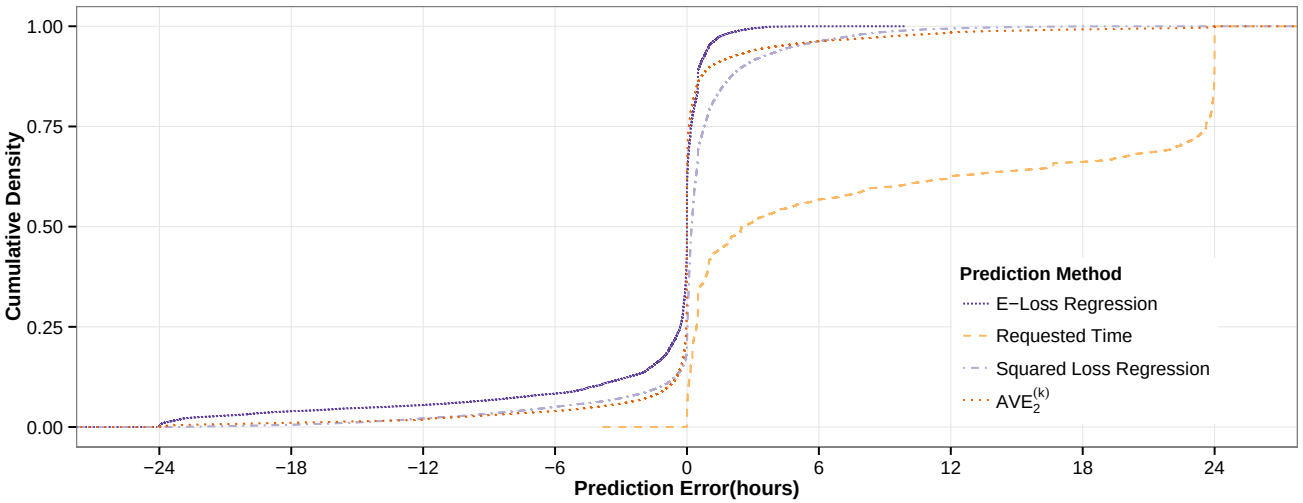

Figure 4: Experimental cumulative distribution functions of prediction errors obtained using the Curie log.

[8] R. Gibbons. A historical application profiler for use by parallel schedulers. In *Job Scheduling Strategies for Parallel Processing*. 1997.

[9] D. A. Lifka. The anl/ibm sp scheduling system. In *Job Scheduling Strategies for Parallel Processing*, 1995.

[10] A. Matsunaga and J. Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *Cluster, Cloud and Grid Computing*, 2010.

[11] LSF (Load Sharing Facility) Features and Documentation. http://www.platform.com/workload-management/high-performance-computing.

[12] Moab Workload Manager Documentation. http://www.adaptivecomputing.com/resources/docs/.

[13] C. Mendes and D. Reed. Integrated compilation and scalability analysis for parallel systems. In *Parallel Architectures and Compilation Techniques*, 1998.

[14] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *Parallel and Distributed Systems*, 2001.

[15] A. Nissimov. Locality and its usage in parallel job runtime distribution modeling using HMM. Master's thesis, The Hebrew University, 2006.

[16] A. Nissimov and D. G. Feitelson. Probabilistic backfilling. In *Job Scheduling Strategies for Parallel Processing*, 2008.

[17] B. Nitzberg, J. M. Schopf, and J. P. Jones. Pbs pro: Grid computing and scheduling attributes. In *Grid resource management*. 2004.

[18] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 1989.

[19] S. Ross, P. Mineiro, and J. Langford. Normalized online learning. *Uncertainty in Artificial Intelligence*, 2013.

[20] J. M. Schopf, F. Berman, J. M. Schopf, and F. Berman. Using stochastic intervals to predict application behavior on contended resources. In *International Symposium on Parallel Architectures, Algorithms, and Networks*, 1999.

[21] W. Smith, I. Foster, and V. Taylor. Predicting application run times with historical information. *Journal of Parallel and Distributed Computing*, 2004.

[22] G. Staples. Torque resource manager. In *Supercomputing*, 2006.

[23] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *Job Scheduling Strategies for Parallel Processing*, 2005.

[24] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. 2007.

[25] A. B. Yoo, M. A. Jette, and M. Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, 2003.