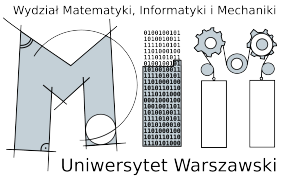

Master of Science in Informatics at Grenoble
Master Mathématiques Informatique - spécialité Informatique
option AIW

Modeling and Estimation for Resource Allocation Problems in Large-Scale Computing Systems

Valentin Reis

June 26 2015

Research project performed at the Institute of Mathematics, Informatics and Mechanics of the University of Warsaw

Under the supervision of:
Krzysztof Rzadca, University of Warsaw

Defended before a jury composed of:
Prof. M.Reza Amini
Prof. C.Berrut
Dr. J.Euzenat
Dr. E.Gaussier

June 2015

Abstract

This report concerns two optimization problems that arise when one tries to allocate resources efficiently in large-scale computing architectures. Firstly, parallel job scheduling in High Performance Computing(HPC) Systems. Then, service allocation in commercial clusters and data-centers.

Both these problems are computationally intractable and subject to uncertainties. Accordingly, they are usually solved using heuristics that provide some form of guarantees on the behavior of the provided solutions. As systems grow in size and storage space becomes cheaper, more and more data are available about their uncertain parameters. For both problems, we ask the following question :

How to use modern modeling and estimation techniques to reduce uncertainty and improve currently used allocation policies?

The report provides a full analysis for the first problem, from design to experimental validation. For the second problem, we focus on the modeling and inferential aspects.

Résumé

Ce rapport de stage concerne deux problèmes d'optimisation survenant d'une volonté d'allouer des resources de manière plus efficace dans les architectures informatiques à large échelle. Premièrement, l'ordonnancement de taches parallèles dans les systèmes de Calcul à Hautes Performances. Puis, l'allocation de services dans les grappes et centre de données commerciaux.

Ces problèmes sont tous deux à la fois intractables et asujettis à des incertitudes. Ainsi, ils sont habituellement résolus à l'aide d'heuristiques pouvant garantir certaines propriétés avantageuses. La taille des systèmes ne cessant d'augmenter, et l'espace de stockage étant toujours plus abordable, il est désormais possible d'obtenir une grande quantités de données concernant les incertitudes. Pour ces deux problèmes, nous nous posons la question suivante :

Comment diminuer les incertitudes à l'aide de techniques modernes de modélisation et d'estimation et ainsi améliorer les politiques d'allocation actuelless ?

Le rapport présente une analyse complète du premier problème, de la conception des méthodes à la validation expérimentale. Dans le traitement du second problème, l'accent est mis sur l'étape de modélisation et d'estimation.

Contents

— 1 —
Introduction

Computer Science's take on optimization has yielded understanding of the hardness of many fundamental problems. Resulting algorithmic techniques, proof methods and problem classes resulting are invaluable as a guide when solving optimization problems in real systems. From a practitionner's point of view, while this fact is fortunate two roadblocks often prevent a successful application:

- The problem may be too different from classic tools and results.
- There may be uncertainties in the description of the problem, e.g the parameters may be known only to belong to a set, or have to be estimated.

While the first issue is solved by doing *more* computer science, the second one is a matter of inference and decision making. There are multiple branches of optimization theory focusing on aspects of uncertainty, among which robust optimization, chance constrained optimization, and stochastic programming. Results from these areas provide many answers when problems are computationally easy, however in some cases the underlying clairvoyant problem – that is, the problem one would have if all parameter realizations were known in advance – is intractable. This issue is amplified by the fact that the stochastic or robust version of an optimization problem is usually[1] *at least as hard* as its clairvoyant version. A classical response practitioners use is to deploy simple heuristics that provide solutions with some form of guarantees on their behavior. While this approach has the double advantage of simplicity and robustness to catastrophic events, it sacrifices potential performance improvements. In resource allocation problems, this can result in many wasted or inactive resources.

This report focuses on two such problems of optimization under uncertainty, namely parallel job scheduling and service allocation. These problems arise when allocating resources in large-scale computing architectures. For each of them, our goal is to provide efficient solutions by modeling uncertainty and performing estimation.

[1] The study of Stochastic Approximation[4] provides a counter-example.

Chapter 2 concerns parallel job scheduling in High Performance Computing (HPC) machines. In this work, we introduced a variant of supervised learning for the purposes of predicting jobs execution times. In particular, we use a linear regression algorithm that is both robust to feature scaling and uses a custom loss function. Then, we modify state-of-the-art scheduling heuristics in order to incorporate the predictions correctly. Finally, we evaluate our approach on five traces from recent HPC systems, and it results in a 28% average performarce improvement with respect to currently used policies. Moreover, as a result of this work, we do not only improve scheduling performance but also empirically determine an appropriate loss function that may be used in subsequent works on runtime prediction for backfilling algorithms.

Chapter 3 concerns service allocation in datacenters. Our focus is the modeling stage of the problem, as there is no currently accepted system model or performance measure for allocation algorithms. Accordingly, our work mostly consists of the analysis of a 500 GB workload released by Google, the first of its kind. This is the first dataset to present CPU usage data on a small time frame (here, the second) on a system so large. This data is rare for good reasons, as it is quite difficult to extract this information without interfering with the system. Perhaps unsurprisingly we uncover important corruption and/or missing values in this dataset. Valentin: elaborate on our added value

Finally, Chapter 4 offers a conclusion on the results and methodology presented in the report, as well as possible extensions.

Full disclaimer about work duration and co-authorship :
Chapter 2 is adapted from a submitted preprint with three co-authors. The work on this subject began at the end of the student's first year internship. The student worked on the project as time allowed until the start of the second year internship, and was finished during the first two months. The work described in Chapter 3 was done during the remainder of the time at the University of Warsaw.

— 2 —

Parallel Scheduling in High Performance Computing

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.

2.1 Introduction

2.1.1 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 [39]. 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.

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

2.1.3 Content

We start by describing the problem in Section 2.2. Then, we present and discuss in Section 2.3 the main existing approaches studied so far in parallel processing field for predicting running times and their use in scheduling. Section 2.4 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 2.5. Section 2.6 reports the simulations done with actual log data. They show that some combinations of prediction and scheduling algorithms improve the system performances.

2.2 Problem description

2.2.1 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 [39].

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 [28] and EASY backfilling [22]. 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 [44]. 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 [26], TORQUE [38], PBSpro [31] or LSF [25] are built upon the same approach with their own specificities. A review on job schedulers can be found in Georgiou [18].

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.

2.2.2 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 2.1 reports the comparison of simulations based on the testbed detailed in Section 2.6. 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, RMSbsld is described in Subsection 2.5.3.

The results reported in Table 2.1 emphasize that clairvoyant simulations are at least 30% better than simulations using the original requested running times. As running

Table 2.1: 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
CTC-SP2	49.6	37.2 (25%)
SDSC-SP2	87.9	70.5 (19%)
KTH-SP2	92.6	71.7 (22%)
SDSC-BLUE	36.5	30.6 (16%)
Curie	202.1	69.9 (65%)
Metacentrum	97.6	81.7 (16%)

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.

2.2.3 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.

2.3 Related Work

2.3.1 Prediction

Historically, job running time prediction has been first attempted [19] 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 [37] 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 [35]. An another example is the method developed by Mendes *et al.* which performs static analysis of the applications [27].

A later survey [24] 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.* [14] 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 [29] 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 [32], and hence it does not use the hypothesis that job submissions would be *i.i.d.*.

2.3.2 Scheduling based on Predictions

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

The Hidden Markov Model used in [29] 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 [30].

The most relevant work for scheduling jobs on large parallel systems using predictions is [40], 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.

2.4 Prediction method

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

2.4.1 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*) [16].

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* [40] 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 [29], 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.

2.4.2 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 2.2. 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 [34], 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:

$$f(\mathbf{w},\mathbf{x}) = \mathbf{w}^\mathsf{T}\Phi(\mathbf{x}) \quad \mathbf{w} \in \mathbb{R}^{1+2n+\binom{n}{2}} \tag{2.1}$$

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

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

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]:

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

[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.

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:

$$\arg\min_{\mathbf{w}} \sum_{j=1}^{N} \mathfrak{L}(\mathbf{x}_j, \mathbf{w}^{\mathsf{T}}\Phi(\mathbf{x}_j), p_j) + \lambda ||\mathbf{w}||_2 \tag{2.2}$$

where λ is the regularization parameter. Once $\mathbf{w}$ has been learned, new running times are predicted through Eq. 2.1.

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 2.1 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.

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

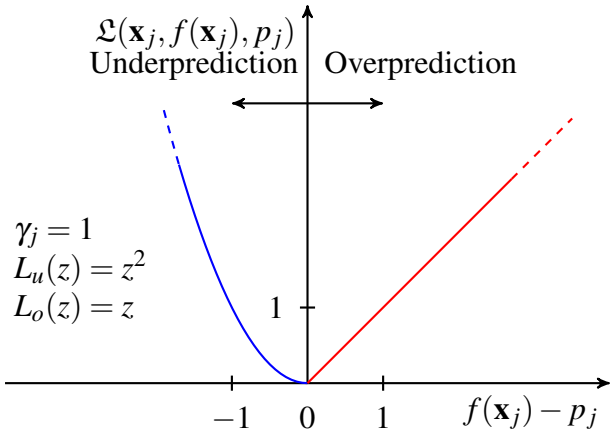

Figure 2.1: Example Loss function $\mathfrak{L}$, plotted with respect to the difference of it's second and third parameters $f(\mathbf{x}_j) - p_j$ (the prediction error).

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 2.3.

Table 2.3: 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.
$5 + \log(\frac{r_j}{p_j})$	Short jobs with large resource request should be well-predicted.
$5 + \log(\frac{p_j}{r_j})$	Long jobs with small resource request should be well-predicted.
$11 + \log(\frac{1}{r_j . p_j})$	Jobs of small "area" should be well-predicted.
$\log(r_j . p_j)$	Jobs of large "area" should be well-predicted.

Finally, for each choice of two basis loss function L_u and 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 [34] algorithm (NAG), a variant of the classical Stochastic Gradient Descent [4]. 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

[3]The notion of *adversary* simply means that the scaling can be arbitrary. See [9] for a comprehensive study.

features are difficult or impossible to normalize (*e.g.* BREAK TIME is unbounded). Section 2.6 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.

2.5 Scheduling

2.5.1 The EASY algorithm

As mentioned in Section 2.2.1, 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 2.2, 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.

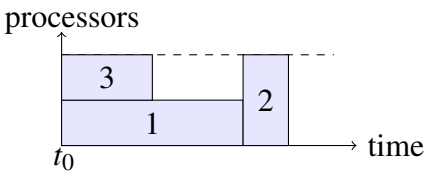

Figure 2.2: Example of EASY for a queue of 3 jobs ordered according to their index. The decision to launch job 1 is taken at t_0. The second ready job (2) can not be executed since there are not enough resources left. Thus, job 3 can also be launched at t_0. The decision to launch job 2 is finally taken when job 1 and 3 complete.

Tsafrir *et al.* proposed in [40] 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.

2.5.2 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 [40], *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.

2.5.3 Objective Functions

As it is commonly admitted [15], the performances of scheduling algorithms are measured using the *bounded slowdown*, which is defined as follows for job j:

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

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:

$$\text{AVEbsld} = \frac{1}{n}\sum_{j}\max\left(\frac{wait_j + p_j}{\max(p_j, \tau)}, 1\right)$$

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

2.6 Experiments

2.6.1 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 2.4: 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

2.6.2 Description of the testbed

We make use in our study of a set of actual workload logs, described in Table 2.4. All these workload logs but Metacentrum are extracted from the Parallel Workload Archive [16]. 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 [17]. 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 2.4 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 2.5.2: REQUESTED TIME, INCREMENTAL and RECURSIVE DOUBLING. Lastly, we rely on the two backfilling algorithms presented in Subsection 2.5.1 , namely EASY and EASY-SJBF.

Table 2.5: 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 2.3 (5 values)

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.* [40].

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

[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

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 2.6.3 outlines the best heuristic triple. Afterwards, Subsection 2.6.4 contains an analysis of the predictions made.

2.6.3 Which heuristic triple is prevalent?

Raw results

Table 2.6: 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
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
KTH-SP2	71.7	49.8	92.6	63.5	62.6 - 93.2	**51.4** - 74.5
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** - 4.01e4
Metacentrum	81.7	67.2	97.6	87.2	86.3 - 98.1	**81.5** - 89.8

As shown in Table 2.6 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 2.6, it is not clear which heuristic triple is prevalent *a priori*.

In particular we observe that the SJBF variant introduced by [40] is rather efficient at leveraging accurate values of the running times, as the CLAIRVOYANT EASY-SJBF algorithm almost always outperforms its competitors.

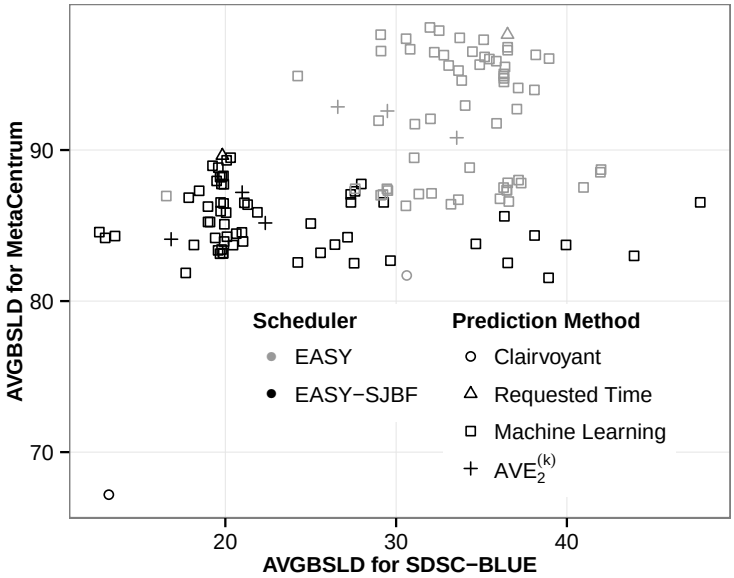

Figure 2.3: Scatter plot of heuristic's relative performance between the KTH-SP2 and SCDC-SP2 logs.

Correlation between logs

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

Figure 2.3 shows the AVEbsld between the CTC-SP2 and SDSC-SP2 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 Pearson'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.

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 2.7 displays the results obtained. 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.

Table 2.7: 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++
CTC-SP2	**20.5** (59%)	49.6	85.8 (-72%)
SDSC-SP2	**75.0** (15%)	87.9	79.4 (10%)
KTH-SP2	**51.4** (44%)	92.6	63.5 (31%)
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%)

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 2.4 with the loss function:

$$\mathfrak{L}(\mathbf{x}_j, f(\mathbf{x}_j), p_j) = \begin{cases} \log(r_j.p_j).(f(\mathbf{x}_j) - p_j)^2 & \text{if } f(\mathbf{x}_j) \geq p_j \\ \log(r_j.p_j).(p_j - f(\mathbf{x}_j)) & \text{if } f(\mathbf{x}_j) < p_j \end{cases} \tag{2.3}$$

Correction mechanism : INCREMENTAL

Backfilling variant : EASY-SJBF

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 [40], as well as a machine learning-based approach with custom loss functions (2.3). We call this loss function E-Loss (for EASY-Loss) and briefly discuss its behavior in the next Subsection.

[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.

2.6.4 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 2.8 which shows the prediction errors of both the $AVE_2^{(k)}(p)$ prediction technique and our E-Loss based approach.

Table 2.8: 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 (2.3) 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 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 2.4 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 2.5 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.

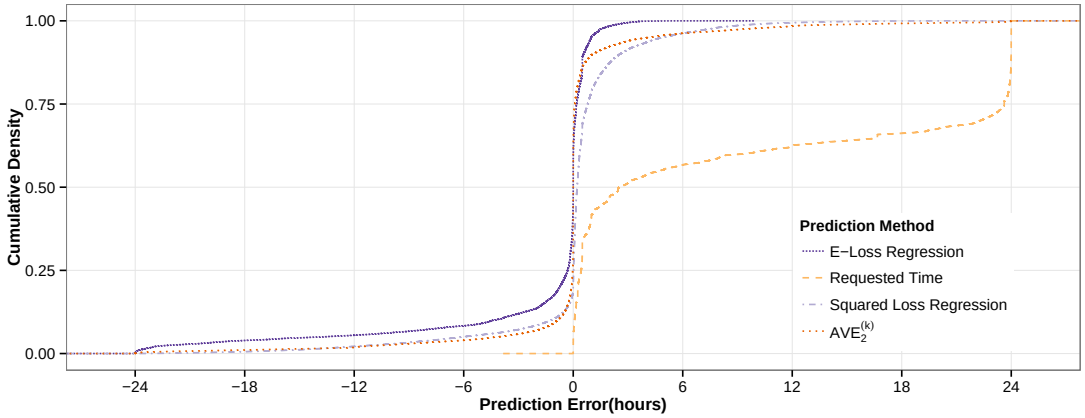

Figure 2.4: Experimental cumulative distribution functions of prediction errors obtained using the Curie log.

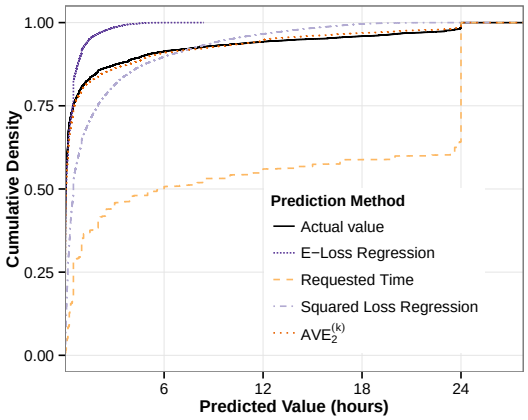

Figure 2.5: Experimental cumulative distribution functions of predicted values obtained using the Curie log.

2.6.5 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 [17]).

2.7 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.

Table 2.2: 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.
$p^{(k)}_{j-1}$	the running time of the last job of the same user, or 0 if such a job does not exist.
$p^{(k)}_{j-2}$	the running time of the second-to-last job of the same user, or 0 if N/A.
$p^{(k)}_{j-3}$	the running time of the third-to-last job of the same user, or 0 if N/A.
$AVE^{(k)}_2(p)$	the average running time of the two last historically recorded jobs of the same user.
$AVE^{(k)}_3(p)$	the average running time of the three last historically recorded jobs of the same user.
$AVE^{(k)}_{all}(p)$	the average running time of all historically recorded jobs of the same user.
q_j	amount of (CPU) resource requested by job j.
$AVE^{(k)}_{hist,r_j}(q)$	average historical resource request of user k, taken at release date of job j.
$\frac{q_j}{AVE^{(k)}_{hist,r_j}(q)}$	amount of resource requested normalized by average resource request.
$AVE^{(k)}_{curr,r_j}(q)$	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)

— 3 —
Service Allocation in Datacenters

3.1 Introduction

Managing computing resources in an efficient manner is crucial to lower the costs, size and energy expenditure of computing systems. While an important body of research concerns HPC, little has been done to allocate resources efficiently in commercial datacenters.

These systems have many differences with traditional HPC. First, tasks may be quite long-running, interacting with users and generating revenue as they go (such tasks may be reffered to as a service). Second, they may have heterogeneous (CPU, Network, Memory) resource usage. Third, this resource usage is quite often lower than machine capacities, and therefore it is possible to run many tasks on a single machine.

As a consequence of the complexity of the problem at hand, there currently is no broadly accepted model on which to start developing service allocation algorithms.

In this chapter, we focus on providing contributions torwards the development of such a model, via the analysis of the month-long trace of a system operated by Google. In particular, we will restrict our analysis on recently added information about the instantaneous CPU usage of long running, latency-sensitive services.

Section 3.2 outlines the reasons behind our approach. Section 3.3 covers related works. Section **??** unfolds our analysis. Section 3.4.2 Concludes on the analysis. The ultimate goal of this project is to provide efficient algorithms for service allocation, and accordingy Section 3.4.2 also lays out the next steps we intend to take in this regard.

3.2 Approach and motivation

An important question for the packing of tasks onto machines is that of whether multiple tasks could be co-allocated on the same machine with over-commitment, which is to say that the usage limits of tasks exceed the machine capacity. Such an over-commitment strategy is already used[**?**] in Google clusters. In this system, services have priority levels (called *appclasses*), which are used to mitigate issues due to the over-commitment.

The strategy is to give resources to services which have the strongest appclass whenever a bottleneck appears. Consequently, priority levels translate to different pricing of resources.

Quality requirements – sometimes called Service Level Objectives (SLO) – may come in multiple forms, such as uptime percentage, or requirements in terms of response time. It is known that tasks in service-oriented clusters use less than their resource ceilings. Some authors report that in Google Compute Cells, 45% of the total CPU power of clusters is unused 99% of the time[8].The natural question is, could these resources be reclaimed? And if so, with what *quality*? While it is clear that such resources are fertile land for besteffort tasks and can always be sold with low guarantees, it is of interest to determine if they can be reclaimed for use with higher quality guarantees (In this chapter, we use the term "High-SLO"). The immediate goal of reclaiming these resources in such a way is to resell them at a higher cost. Note that as long the SLO of services determine their precedence in the use of the resources, one can always give local priority to high-SLO services over low-SLO services if a resource is under-provisionned. Consequently, the question of estimating the high-quality reclaimability of resources reduces to:

Can multiple High-SLO services be over-committed to a machine while ensuring good resource quality with high confidence?

In this chapter, we contribute to answering this main question. In particular, we concern ourselves with the CPU usage of high-priority services. As these services may have high response times, we focus our analysis on a trace that was recently re-released by Google, the first to contain CPU information on a small time scale, that of one second.

Our analysis is based on the idea that by modeling the behavior of high-SLO tasks, one might be able to provide additional good quality resources for them. The next section covers the related works pertaining to the analysis of the Google trace.

3.3 Related Work

While datacenter systems always increase in size, few companies actually operate such systems. Still, reports such as [**?**]have been made are available from large-scale datacenter operators. Not only do these authors provide details about the working of their system, they also provide a dataset of its usage[42], although the data is partially obfuscated[**?**] due to business prerogatives.

Similar work has outlined the necessity of developing an efficient model for service allocation and data centers. In particular, [8] emphasises to the difficulty of evaluating a packing of services onto machines.

The Google Trace[42] is a month-long dataset describing the resource usage of more than 25 million tasks over 12.500 hosts. It has been extensively used both in terms of analysis[33, 12, 1, 23, 10] and usage[21, 3, 7, 36, 5, 46, 13, 20, 45, 2, 11, 41].

The dataset has a total size of 500 GB when encoded in ASCII. It contains in-depth information about the services, the machines, and the historical events that happened during the month the monitoring took place.

This data set was recently[43] enhanced by the addition of more involved CPU usage measurements.

The next section describes this dataset and characterizes it.

3.4 Analysis of the instantaneous CPU usage

CPU usage information in the Google Trace has the following format.

The services are structured via *job* and *task* layers. A *job* contains multiple *tasks*.

As mentionned previously, this section restricts the analysis to tasks which have high SLO. In the google trace, there are two fields relating to the relative importance of tasks. On one hand, the task *priority*, which determines precedence of the task in the scheduling queue. The second field is the *scheduling class*, which determines the precedence of the task when trying to obtain resources locally, in competition with neighboring tasks. In other words, while the task priority determines how fast a task will be served resources, the scheduling class determines the quality of these resources. Indeed, in this system, if a task has the highest scheduling class on a machine, it is granted the full usage of the machine in case of any resource bottleneck.

CPU task usage is measured every 1 second. For every 5 minute period of activity of any task on the system, the mean μ_{5s} and maximum $\max_{5s}$ values of these measurements are reported. In addition, for each of these 5 minute periods, a randomly selected 1 second measurement μ_{1s} is reported.

As mentionned previously, we restrict our analysis to High-SLO tasks. With this dataset, this means filtering tasks based on their scheduling class attribute. This attribute has three levels $1, 2, 3$ After discussion with the authors of the dataset, we determined that levels 2 and 3 were appropriate to consider as tasks with a high SLO. Indeed, tasks of scheduling class 2 and 3 constitute mostly user-based services, and consequently they are not only revenue generating but also have low latency requirements. Additionally, we focus our analysis on month-long running jobs. Indeed, for long-running jobs not only are more samples available, but additionally it is possible to hope that whichever characteristics of tasks we will uncover could be predicted using historical data.

3.4.1 Filtering out corrupted values

No dataset is perfect, and the first step of the analysis is the filtering of corrupted values. Figure 3.1 shows how the task "true" mean (denoted by $AVG_{\text{task}}(\mu_{5s})$) correlates with the task "reconstructed" mean (denoted by $AVG_{\text{task}}(\mu_{1s})$), for all long running, high class tasks.

One can see from this graph that the average CPU sampled usage for each tasks is either close to the average mean usage, or close to zero. To be more precise, there

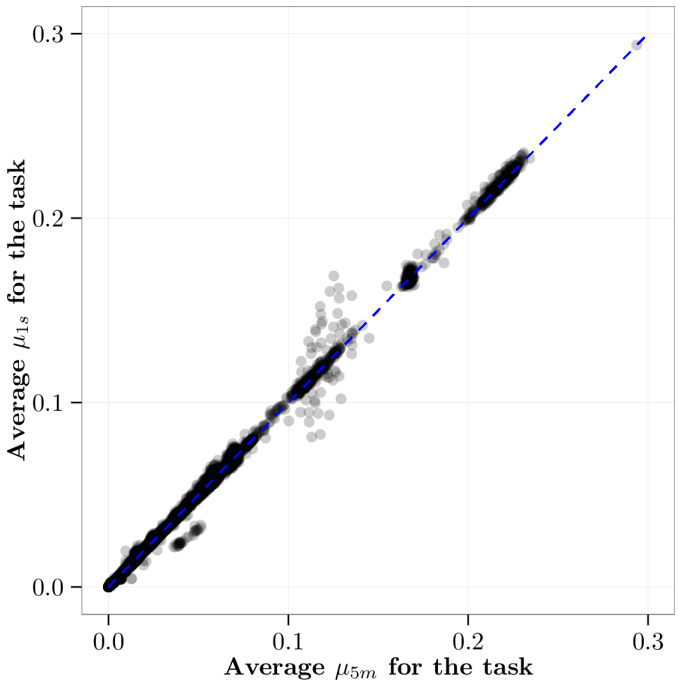

Figure 3.1: Scatterplot of the mean values of task's CPU usage obtained via 5 minute and 1 second samples.

are Valentin: x tasks with a mean sampled CPU usage of exactly zero, among all the Valentin: x tasks of the trace. The bi-modal nature of this graph entails two possible explanations: Tasks for which μ_{1s} is always zero might either not be measured (data corruption) or they might be have an extremely bursty CPU utilization that is not captured by the sample rate.

In order to discriminate between these two hypotheses, we note that the number of tasks which have exactly 1, 2, 3, and 4 non-null CPU sample measurements are respectively x,y,z, and u. Additionally, the median number of measurements per task is Valentin: x. We conclude without further analysis that these zero values in the sampled CPU usage data are due to corruption in the measurement process.

Concluding on data corrupution, we notice that the presence of sampled CPU usage in a task depends on the job ID. To be more specific, for each job either all its tasks for which $AVG_{\text{task}}(\mu_{5m}) \neq 0$ have at least a non-zero μ_{1s} measurement, or all its tasks have $\mu_{1s} = 0$. This indicates that measurement of the sample CPU usage was activated on a per-job basis. We restrict further analysis to the tasks belonging to the Valentin: x jobs for which at least one μ_{1s} measurement is not null and refer to them as μ_{1s}-measured jobs.

3.4.2 Volatility of instantaneous CPU usage

The first question we ask regarding the over-subscription of multiple tasks on one machine concerns the stationarity of the variability of the CPU usage of tasks. For a specific task one should ask whether the variability of μ_{1s} stays the same, or changes as time passes.

Figure 3.2 shows the empirical cumulative distribution functions of the task-wise coefficients of variations for both μ_{1s} and μ_{5m}, for all tasks belonging to μ_{1s}-measured jobs. We recall the value of the coefficient of variation for the measurements of μ_x associated with a task:

Valentin: CV vs median based, explain difference tradeoff

$$CV_{\text{task}}(\mu_x) = \frac{\sigma_{\text{task}}(\mu_x)}{AVG_{\text{task}}(\mu_x)}$$

Where $\sigma_{\text{task}}(\mu_x)$ is the sample standard deviation of μ_x for this task.

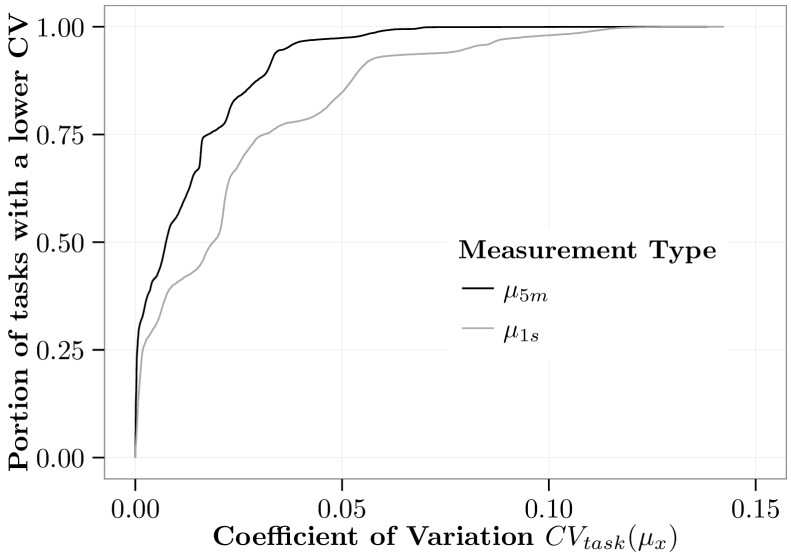

Figure 3.2: ECDF of the coefficients of variations of 5 minutes-based mean and sampled CPU usage for all tasks.

This graph motivates further the study of the instantaneous CPU usage. Indeed, it confirms that μ_{1s} is more volatile than μ_{5m}. Valentin: quantitative comment.

When comparing μ_{1s} and μ_{5m}, one may ask by how much does μ_{1s} deviate from μ_{5m}. Figure 3.3 shows the histogram of the average normalized absolute difference between instantaneous and 5 minute averaged CPU usage for all task belonging to a μ_{1s}-measured job. In other terms, Figure 3.3 shows the histogram of $AVG_{\text{task}}(|d_{1,5}|)$ for all tasks belonging to a μ_{1s}-measured job, where :

Valentin: add median,comment

$$d_{1,5} = \frac{\mu_{5\text{mn}} - \mu_{1\text{s}}}{\mu_{5\text{mn}}}$$

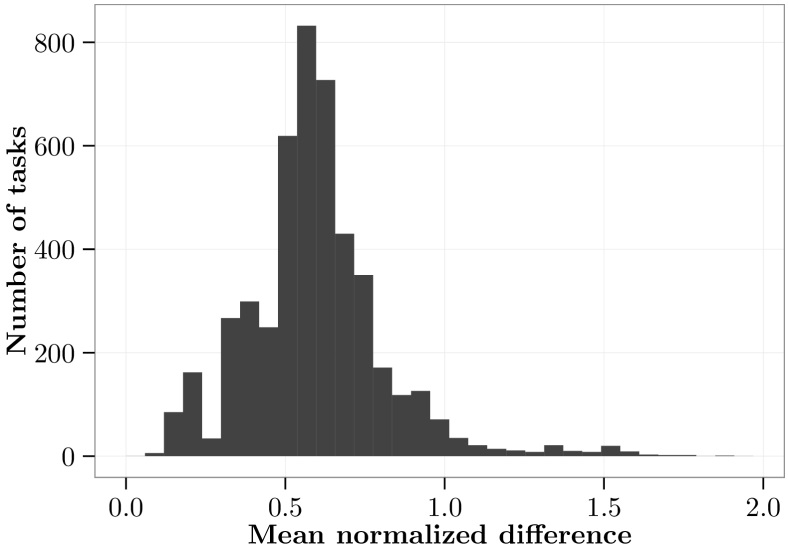

Figure 3.3: Histogram of the normalized deviation of CPU usage for all tasks.

This graph indicates that the 5 minute-based CPU usage mean is not a good surrogate for estimating instantaneous CPU usage. Indeed, the values are distributed around their average of Valentin: x, which means that an average error of Valentin: x% is made when using 5 minute means to estimate instantaneous CPU usage.

Additionally, Figure 3.4 shows the empirical cumulative distribution function of $d_{1,5}$ for all measurement periods belonging to any μ_{1s}-measured job. Valentin: quantitative comment

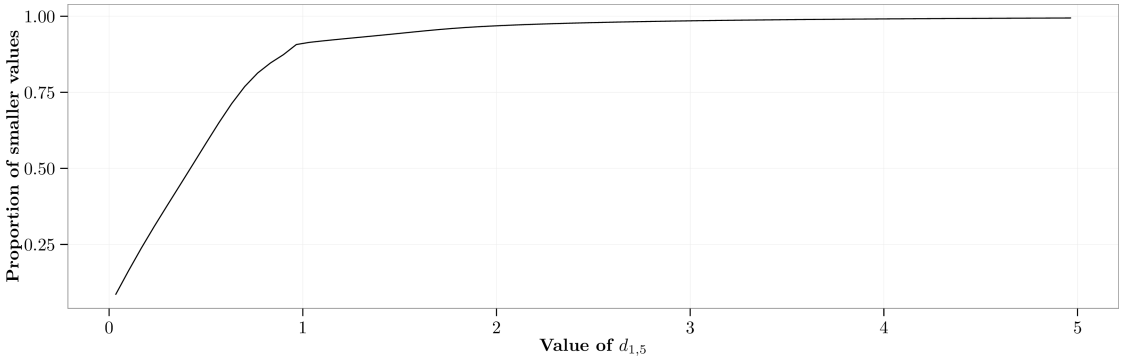

Figure 3.4: ECDF of the normalized deviation $d_{1,5}$ for all measurements.

Figure 3.5 shows the heatmap of the normalized 5 minutes CPU usage averages against the normalized 1 second samples. Normalization is performed using the CPU resource request CPU_{req} of the task being measured. Valentin: comment

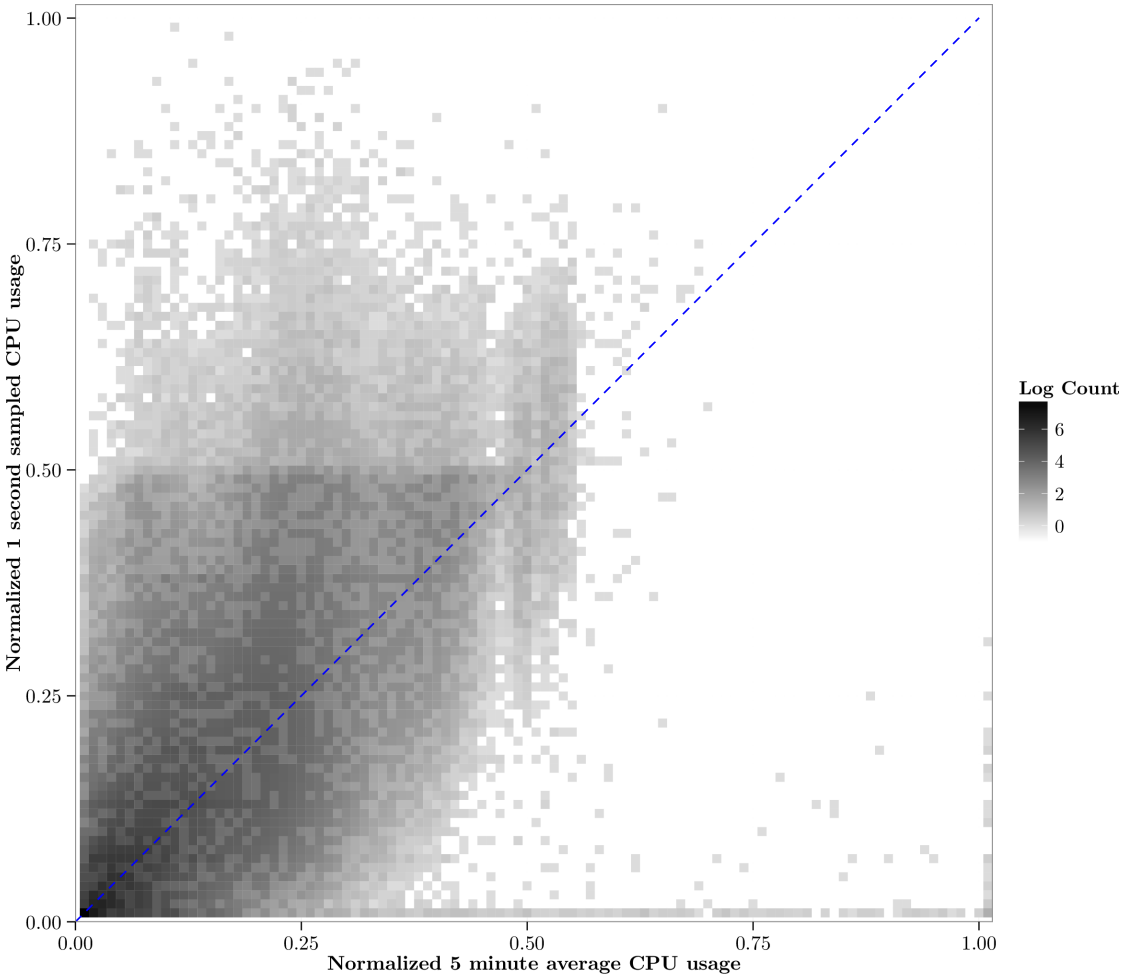

Figure 3.5: Heatmap of $\frac{\mu_{5\text{mn}}}{CPU_{req}}$ against $\frac{\mu_{1\text{s}}}{CPU_{req}}$ for all measurements in the trace. The color scale is logarithmic of base 10 in the number of measurements in each bin

— 4 —

Conclusion

This report has explored ways in which modeling and estimation can be used to improve resource allocation policies in large scale computing systems.

In the case of parallel job scheduling in HPC systems, we presented an approach based on supervised learning. We modified the existing heuristics to incorporate predictions and determined experimentally an appropriate loss function for the learning algorithm. This approach is successful, as it results in an average performance gain of 28% compared to the currently used algorithm.

In the case of service allocation in data-centers, we focused on the modeling aspect. To that end, we studied the first available large dataset describing the "instantaneous" CPU usage of tasks on a large system. The 500 GB dataset contains information about CPU use at the scale of the second. The rarity of this information is tied to the difficulty of extracting it, and accordingly we uncover strong corruption in this data. The analysis then focuses on quantifying the importance of instantaneous CPU usage and concludes that for some tasks, this new information not only must be taken into account in the model, but can effectively be incorporated to algorithms using simple predictive techniques. Valentin: elaborer

Multiple possible specialized extensions to each problem are mentioned in their respective chapters. Here, we instead focus on the insight gained from both applications and possibilities for more fundamental work.

A first remark concerns the methodology used. Here, we take the approach of incorporating statistical estimation techniques into existing algorithms. It could be rewarding to take the more "declarative" approach of stating the full optimization problems (including models of uncertainty and necessary functional requirements – such as the absence of starvation for a scheduling policy), before trying to find algorithms. This could lead to two interesting kinds of results.

On one hand, it could allow to develop bounds similar to those from statistical learning theory to provide guarantees. Bounds from statistical learning typically concern sum of random variables for simple stochastic programming problems. When formulating optimization problems with probabilistic models of uncertainty, one often uses measures the objective function that go beyond the expectation, such as the variance. This is

not a barrier however, as in the last decade many new tools from concentration inequalities have been brought up that allow to bound the tail probabilities of various functions of random variables.

Additionally, once a problem is fully formulated with the model of uncertainties, it could be possible to look for a solution that is *easier* than the clairvoyant counterpart. As demonstrated by the success of Stochastic Approximation, there are cases where the uncertainties in a problem make over-optimization useless. Quantifying this phenomenon could lead to efficient and tractable algorithms. We briefly expand on the term of Stochastic Approximation (SA), as it is sometimes confounding for algorithmicists. SA does not refer to approximation algorithms. It is a numerical optimization idea for stochastic programming in which sampling of the uncertain parameters is used directly to design randomized optimization algorithms. Again, note that stochastic programming is not to be confused with stochastic optimization, *a.k.a* randomized algorithms for deterministic problems, *i.e.* quicksort.

The second possible extension lies in introducing learning-based approaches for leveraging currently existing solutions. In other words, we propose to develop methods of algorithm portfolios, in which already existing heuristics or algorithms are combined (e.g. with a majority vote) in order to provide decisions. There are multiple ways to do so, depending on the hypothesis about uncertainties and the type of guarantees that are deemed necessary on the performance of the systems. In particular, the frameworks of prediction with expert advice [9] and online hyper-heuristics[6] could be used.

Bibliography

[1] `http://www.adaptivecomputing.com/resources/docs/`. [Online; accessed 20-May-2015].

[2] Omar Arif Abdul-Rahman and Kento Aida. Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 272–277, Singapore, December 2014.

[3] M. Amoretti, A.L. Lafuente, and S. Sebastio. A cooperative approach for distributed task execution in autonomic clouds. In *21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 274–281, Belfast, UK, February 2013. IEEE.

[4] Alkida Balliu, Dennis Olivetti, Ozalp Babaoglu, Moreno Marzolla, and Alina Sirbu. BiDAl: Big Data Analyzer for cluster traces. In *Informatik Workshop on System Software Support for Big Data (BigSys)*. GI-Edition Lecture Notes in Informatics, September 2014.

[5] Léon Bottou. Stochastic learning. In *Advanced lectures on machine learning*, pages 146–168. Springer, 2004.

[6] David Breitgand, Zvi Dubitzky, Amir Epstein, Oshrit Feder, Alex Glikson, Inbar Shapira, and Giovanni Toffetti. An adaptive utilization accelerator for virtualized environments. In *International Conference on Cloud Engineering (IC2E)*, pages 165–174, Boston, MA, USA, March 2014.

[7] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. 2010.

[8] Faruk Caglar and Aniruddha Gokhale. iOverbook: intelligent resource-overbooking to support soft real-time applications in the cloud. In *7th IEEE Inter-*

national Conference on Cloud Computing (IEEE CLOUD), Anchorage, AK, USA, Jun–Jul 2014.

[9] Marcus Carvalho, Walfredo Cirne, Franciso Brasileiro, and John Wilkes. Long-term slos for reclaimed cloud computing resources. In *ACM Symposium on Cloud Computing (SoCC)*, pages 20:1–20:13, Seattle, WA, USA, 2014.

[10] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA, 2006.

[11] Sheng Di, Derrick Kondo, and Walfredo Cirne. Characterization and comparison of cloud versus Grid workloads. In *International Conference on Cluster Computing (IEEE CLUSTER)*, pages 230–238, Beijing, China, September 2012.

[12] Sheng Di, Derrick Kondo, and Walfredo Cirne. Host load prediction in a Google compute cloud with a Bayesian model. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 21:1–21:11, Salt Lake City, UT, USA, November 2012. IEEE Computer Society Press.

[13] Sheng Di, Derrick Kondo, and Cappello Franck. Characterizing cloud applications on a Google data center. In *42nd International Conference on Parallel Processing (ICPP)*, Lyon, France, October 2013.

[14] Sheng Di, Yves Robert, Frédéric Vivien, Derrick Kondo, Cho-Li Wang, and Franck Cappello. Optimization of cloud task processing with checkpoint-restart mechanism. In *25th International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Denver, CO, USA, November 2013.

[15] Rubing Duan, F. Nadeem, Jie Wang, Yun 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.

[16] Dror G. Feitelson. Metrics for parallel job scheduling and their convergence. In DrorG. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 188–205. Springer Berlin Heidelberg, 2001.

[17] Dror G. Feitelson, Dan Tsafrir, and David Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74(10):2967 – 2982, 2014.

[18] Eitan Frachtenberg and Dror G. Feitelson. Pitfalls in parallel job scheduling evaluation. In *Job Scheduling Strategies for Parallel Processing*, pages 257–282, Berlin, Heidelberg, 2005. Springer-Verlag.

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

[20] Richard Gibbons. A historical application profiler for use by parallel schedulers. In DrorG. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 58–77. Springer Berlin Heidelberg, 1997.

[21] Qiang Guan and Song Fu. Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In *32nd IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 205–214, Braga, Portugal, September 2013.

[22] Jesus Omana Iglesias, Liam Murphy Lero, Milan De Cauwer, Deepak Mehta, and Barry O'Sullivan. A methodology for online consolidation of tasks through more accurate resource estimations. In *IEEE/ACM Intl. Conf. on Utility and Cloud Computing (UCC)*, London, UK, December 2014.

[23] David A Lifka. The anl/ibm sp scheduling system. In *Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer, 1995.

[24] Zitao Liu and Sangyeun Cho. Characterizing machines and workloads on a Google cluster. In *8th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS)*, Pittsburgh, PA, USA, September 2012.

[25] 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.

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

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

[28] Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *Parallel and Distributed Systems*, 12(6):529–543, June 2001.

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

[30] Avi Nissimov and Dror G. Feitelson. Probabilistic backfilling. In *Job Scheduling Strategies for Parallel Processing*, pages 102–115, Berlin, Heidelberg, 2008. Springer-Verlag.

[31] Bill Nitzberg, Jennifer M Schopf, and James Patton Jones. Pbs pro: Grid computing and scheduling attributes. In *Grid resource management*, pages 183–190. Springer, 2004.

[32] Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[33] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM Symposium on Cloud Computing (SoCC)*, San Jose, CA, USA, October 2012.

[34] Stéphane Ross, Paul Mineiro, and John Langford. Normalized online learning. *Uncertainty in Artificial Intelligence*, abs/1305.6646, 2013.

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

[36] Stefano Sebastio, Michele Amoretti, and Alberto Lluch Lafuente. A computational field framework for collaborative task execution in volunteer clouds. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 105–114, Hyderabad, India, June 2014. ACM.

[37] Warren Smith, Ian Foster, and Valerie Taylor. Predicting application run times with historical information. *Journal of Parallel and Distributed Computing*, 64(9):1007–1016, September 2004.

[38] Garrick Staples. Torque resource manager. In *Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[39] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. Modeling user runtime estimates. In *Job Scheduling Strategies for Parallel Processing*, JSSPP'05, pages 1–35, Berlin, Heidelberg, 2005. Springer-Verlag.

[40] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. In *Parallel and Distributed Systems*, volume 18, pages 789–803. IEEE, 2007.

[41] Guanying Wang, Ali R. Butt, Henry Monti, and Karan Gupta. Towards synthesizing realistic workload traces for studying the Hadoop ecosystem. In *19th IEEE Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 400–408, Raffles Hotel, Singapore, July 2011. IEEE Computer Society.

[42] John Wilkes. More Google cluster data, `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`. Google research blog, November 2011. [Online; accessed 20-May-2015].

[43] John Wilkes and Charles Reiss. Google cluster data 1022-2. `http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_2`, 2014. [Online; accessed 20-May-2015].

[44] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.

[45] Qi Zhang, Mohamed Faten Zhani, Raouf Boutaba, and Joseph L. Hellerstein. HARMONY: dynamic heterogeneity-aware resource provisioning in the cloud. *The 33rd International Conference on Distributed Computing Systems (ICDCS)*, pages 510–519, July 2013.

[46] Qi Zhang, Mohamed Faten Zhani, Raouf Boutaba, and Joseph L Hellerstein. Dynamic heterogeneity-aware resource provisioning in the cloud. *IEEE Transactions on Cloud Computing (TCC)*, 2(1), March 2014.