

Towards High-Level SLAs with Heterogeneous Workloads: Job Resource Requirements Prediction for Deadline Schedulers

Gemma Reig¹, Javier Alonso² and Jordi Guitart³

Abstract—When executing their tasks, Grid and Cloud users want to express their requirements in terms of high-level metrics (e.g. in terms of execution time, not in terms of CPU MHz). Moreover, at the submission time they would like to know if the resource provider will fulfil with their requirements in order to decide if they would rather prefer another provider. On the other hand, the resource provider have to translate these high-level metrics into hardware related metrics, to know if he have enough resources to execute the user's requests. In this context, we present our prediction system to foresee the amount of CPU required for a job to finish before its deadline. This prediction system uses machine learning techniques to learn about the jobs and online adjust itself. Before all this training is done, the Prediction System uses an analytical model for this purpose. We also contribute with a deadline-based scheduler which uses these predictions to discard jobs that will not meet its deadline in order to maximize the provider's revenue by means of a dynamic and efficient resource allocation to jobs. We show how our system is able to provide higher revenue to resource providers compared to simple yet well known schedulers like EDF, SJF, etc.

Key words— SLA, heterogeneous workloads, resource requirements prediction.

I. INTRODUCTION

USERS of modern distributed systems like the Grid or Cloud want to execute their applications without concerning in hardware related metrics. This way, users will is to negotiate with the resource provider in high-level terms of quality of service (e.g. in terms of execution time, not in terms of CPU MHz), that is, the quality of their experience [1]. On the other hand, the provider's concern is about business metrics, such as its revenue. Therefore, the resource provider should translate somehow these high-level metrics to resource requirements. It is so because the provider needs to know the amount of resources required by the application to be able to meet the agreed high-level metric, charging for its execution. These metrics are collected in an agreement between the user and the provider called SLA (Service Level Agreement) which specifies the user's

requirements (job deadline, response time, etc.), the price if the agreement is fulfilled and the penalty if the terms are not met.

Grid and Cloud applications exhibit very heterogeneous nature, ranging from HPC jobs to web services. In addition, providers' workloads are typically highly dynamic. This makes not assumable to statically provision dedicated resources to the applications, since this lead to low resource utilization. In order to be profitable, service providers tend to share their resources among multiple concurrent applications owned by different customers. Resource sharing implies the cohabitation of services with very different behaviors and requirements. However, the provider must guarantee that each application has always enough resources to meet the agreed performance goals. The key is a smart utilization of the resources with the objective of maximize the provider's revenue.

One key technology in this context is virtualization. Its isolation property may be used by providers to dynamically allocate unused resources left by web applications to execute batch jobs. Current broadly used schedulers do not take advantage of virtualization properties as they use to allocate the whole resource to a single task. It is so because they do not know in advance the minimum amount of resources required to meet task's SLA. Using prediction techniques to determine the task's minimum resource requirements to fulfil the SLA, the provider is able to discard jobs in advance, avoiding the risk of wasting its resources in executing jobs that will not meet their deadlines, penalizing provider's revenue.

The contribution of this paper consists of a prediction system to determine the minimum job resource requirements to execute a job before its deadline. It consists of two prediction modules. First, an Analytical Predictor that uses a mathematical expression to make the predictions and thus does not need training. Second, a Self-Adjusting Predictor that uses machine learning to learn from previous executions of tasks and make predictions. We also show how different schedulers may take advantage of this prediction in order to maximize the provider's revenue.

The reminder of this paper is structured as follows: in Section II we present the scenario where our proposal applies, Section III shows the system architecture, Sections IV and V present respectively the Prediction System and the Scheduler, in Section VI we present the results of the system, the related work is explained in Section VII and finally in Section VIII

¹Dpto. de Arquitectura de Computadores, Univ. Politècnica de Catalunya, e-mail: greig@ac.upc.edu and Autonomic Systems and e-Business Platforms, Barcelona Supercomputing Center, e-mail: gemma.reig@bsc.es

²Dpto. de Arquitectura de Computadores, Univ. Politècnica de Catalunya, e-mail: alonso@ac.upc.edu and Autonomic Systems and e-Business Platforms, Barcelona Supercomputing Center, e-mail: javier.alonso@bsc.es

³Dpto. de Arquitectura de Computadores, Univ. Politècnica de Catalunya, e-mail: jguitart@ac.upc.edu and Autonomic Systems and e-Business Platforms, Barcelona Supercomputing Center, e-mail: jordi.guitart@bsc.es

we conclude with our future work and conclusions.

II. SYSTEM MODEL

Our proposal targets Grid and Cloud resource providers that use virtualization techniques in their hardware resources. In this context, providers should handle heterogeneous and dynamic workloads. These workloads consist of both web applications and batch jobs.

For the scope of this paper we focus in providing resource predictions for repetitive CPU intensive batch jobs as they are common in these environments. Memory or data intensive jobs as well as the prediction for the web applications are part of our future work. However, we assume the existence of a predictor that provides us with the CPU requirements of the web applications over time.

III. ARCHITECTURE

This section presents the logical architecture of the system (see Figure 1). This is composed by three main components:

- The Scheduler accepts incoming jobs and web requests to be planned. Once a job arrives, it waits until the Policy module queries the Prediction System and Resource Status module and decides whether it can start executing, it should be discarded or its start or discard should be delayed, depending on the policy being used. It is the responsible of efficiently allocate resources to jobs and elastically size up the resource allocation for the web applications.
- The Prediction System is in charge to predict the minimum resource requirements needed to meet SLAs. It consists of an Analytical Predictor module that can answer queries without being trained and a Self-Adjusting Predictor module that needs to be trained in order to answer the same queries. Besides, it is able to learn about its errors during execution.
- Physical resources where jobs and web applications are placed.

IV. RESOURCE REQUIREMENTS PREDICTION

The Prediction System stores and manages information of previous job executions in order to predict the amount of required resources by a task. It needs the user to provide an initial information about the duration of the tasks assuming a whole CPU for its execution. This information is just required once per application, because the Analytical Predictor is able to predict with just one previous information sample.

We have used the Java Grande Benchmarks FFT and Montecarlo and the Mencoder applications on a Xen virtual machine with 1GB of memory and an Intel(R) Xeon(TM) MP CPU 3.16GHz to train and test the Prediction System. We have used these applications because they are all CPU-intensive.

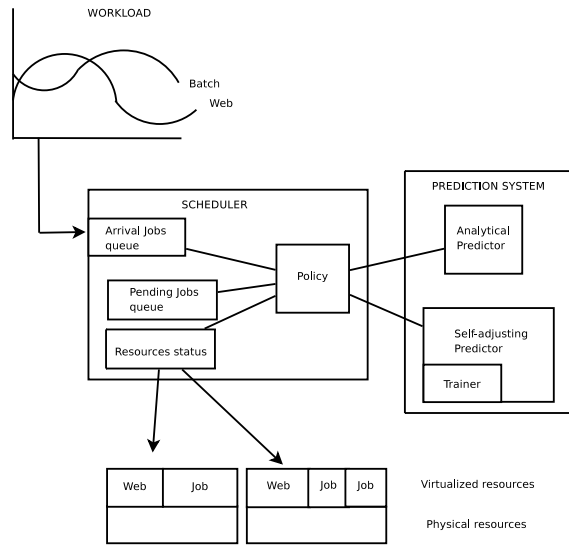


Fig. 1. System architecture.

A. Analytical Predictor

The aim of the Analytical Predictor is to answer predictions while the Self-Adjusting Predictor is not trained. We have observed empirically that CPU-bounded jobs can be approximated by the following equation:

$$T_{exec} = \frac{k}{\%CPU} \quad (1)$$

where T_{exec} is the execution time of the job assuming that a CPU share of $\%CPU$ is allocated to the job and $k = T * \%CPU$, where the execution time (T) and the CPU share are obtained from the most similar (in terms of execution time) past executions or from the information given by the user. Figure 2, which compares the real and the analytically predicted execution time, shows that Equation 1 can closely approximate the execution time for the three applications.

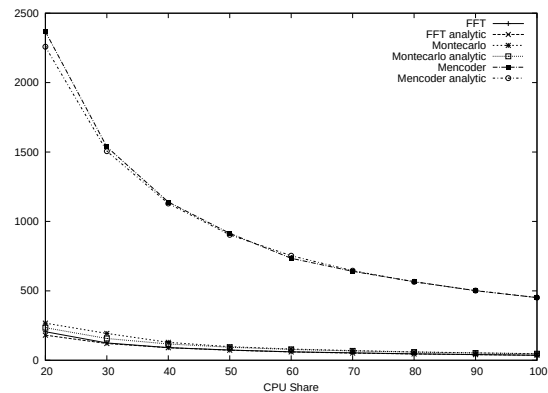


Fig. 2. Comparison among analytical predictions and real executions

Therefore, the Analytical Predictor module uses Equation 2 to determine the minimum CPU requirements.

$$\%CPU = \frac{k}{T_{exec}} \quad (2)$$

B. Self-Adjusting Predictor

The Self-Adjusting Predictor uses machine learning techniques to predict the CPU share required to execute a job before its deadline expires. The predictor is trained with sets of data from previous executions of tasks containing: *the application name, the CPU share, and the execution time*. Throughout Section VI, we evaluate the following two machine learning techniques to perform this prediction:

Simple Linear Regression: is a type of regression that computes the linear relationship between the dependant variable ($1/T_{exec}$ in this case) and the independent variable %CPU. This regression computes the line that minimizes the sum of the squares of the vertical distances of the input samples from the line.

M5: consists of a decision tree in which each leaf is a regression model itself [2].

We have chosen these two techniques because their prediction accuracy is high as we will see later on the evaluation section and because their computational complexity is very low which introduces a low overhead in an online system.

V. SCHEDULER

In this section, we describe a Scheduler that takes profit of the prediction mechanisms in order to improve the provider's revenue. The Scheduler uses the prediction obtained by the Prediction System to know in advance if it is able to meet the job deadline as it is also known the resources available. This way, providers can make the decision of discarding a job if they predict that they will not be able to execute it on time to gain any or almost any revenue. The provider can also delay this decision along the lifetime of the job (e.g. the provider may accept a job and then discard it if a new job with highest revenue arrives at the system and he is not able to fulfil both SLAs). This decision is consequence of the policy being used. The provider can choose any policy from the most classic policies such as FCFS (First Come First Serve), SJF (Shortest Job First), etc. to those policies that incorporate business elements in them such as HV (Highest Value) which prioritize those jobs with highest revenue, HD (Highest Density) which prioritizes those jobs with highest ratio revenue per remaining execution time, etc. Moreover, for each of these policies there exist the non preemptive version, meaning that when a job starts its execution does not stop executing until its execution is completed or the preemptive version, meaning that when a new job has higher priority than one that is running it is paused in favour of the new one.

Regarding users, they are also favoured because they know earlier if the provider will fulfil or not the SLA. This way they can search for another provider to execute the job or renegotiate the SLA terms.

Although the scheduler knows the required amount of resources to execute a job, it allocates the maximum of available resources. It is so because,

as we have assumed that jobs are CPU-bounded, the job will make more progress than the strictly required to meet the deadline. Therefore, it will finish before its deadline. This way, we are fully using the resources when the system is underloaded in order to have more resources available for future overloaded periods.

VI. EXPERIMENTS AND RESULTS

So far, we have presented our proposal. In this section we are going to evaluate it. First, we show that the Analytical Predictor module is useful for discarding unfeasible jobs and then we evaluate the accuracy of the Self-Adjusting Predictor module using two different algorithms to predict. Finally, we compare these two prediction modules.

A. Prediction Usefulness

The aim of this test is to demonstrate empirically that the Prediction System helps schedulers to efficiently allocate resources to jobs whatever the policy used is.

A.1 Experimental setup

We simulate our system using the Analytical Predictor with 10 simulated CPUs and running 25000 jobs of the Grid workload of Grid'5000 [8]. We randomly have set the revenue for each job in the range [0,1) and have supposed a periodic dynamic occupation of the resources due to the web workload. Specifically, the average web occupation during a period $[t_0, t_1]$ is computed by the following formula:

$$\frac{\int_{t_0}^{t_1} (\sin(\frac{x}{k}) + 1) * \frac{Max-Min}{2} + Min dx}{t_1 - t_0} \quad (3)$$

This formula gives a web sinus-periodic workload between a Max and Min values. We set the k factor to 10000 to avoid sudden changes in the workload and we set Max and Min values to 100% and 0% respectively.

A.2 Results

Figure 3 shows the revenue gained by a provider using a set of policies with and without prediction. Notice, that the prediction system helps providers to increase its revenue whatever policy they use, whether it is preemptive (a job that is executing can be resumed and enter in the queue again in favor of a more priority job) or not. To complement it, Figure 4 show the number of jobs being discarded and the number of jobs being violated (i.e. breaking the SLA) for each policy. Notice that with prediction, providers do not violate any SLA, thus they charge for all the jobs they execute without suffering any penalty. They discard the jobs if they realize that SLA cannot be accomplished. Policies without using prediction only discard jobs when its deadline is in the past.

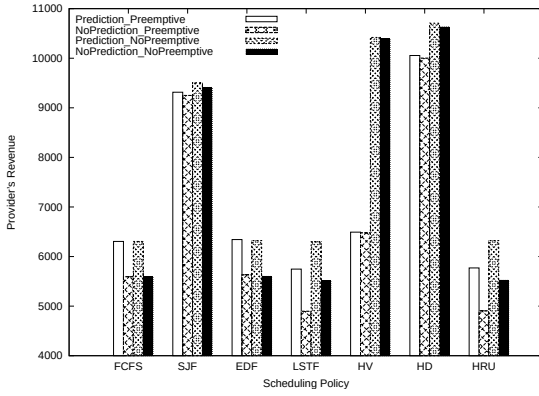


Fig. 3. Providers' benefit using prediction. As jobs run in a virtualized environment we set the minimum CPU requirement to 20%. Policies used: FCFS (First Come First Serve), SJF (Shortest Job First), EDF (Earliest Deadline First), LSTF (Least Slack Time First), HV (Highest Value), HD (Highest Density), HRU (Highest Reward and Urgency) [3]

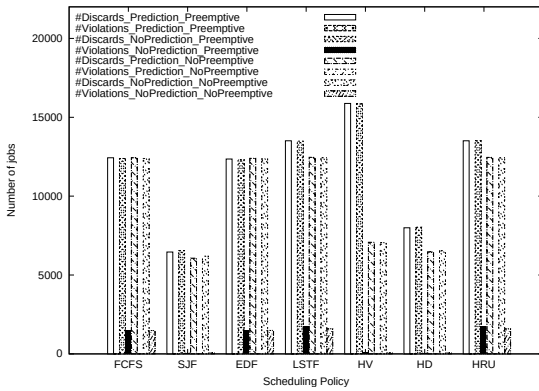


Fig. 4. Number of jobs discarded and SLA violations with and without using prediction.

B. Self-Adjusting Predictor accuracy

In this section we show the accuracy of the predictions performed using the two machine learning techniques described in Section III. We have used two different sets of data: one for the training phase and one, totally different, to evaluate the accuracy of the generated models produced during the learning process. Figure 5 compares the real CPU percentage required to meet the deadline with the predicted CPU percentage by M5 algorithm for the applications FFT, Montecarlo and Mencoder. Only small differences are appreciable among the CPU predicted and the real CPU required using the M5 algorithm and trained with samples consisting of the application name, CPU share and $1/T_{exec}$.

On the other hand we also consider and study the possibility of predicting each application using a linear regression per each one. Table I summarizes the obtained results showing that the linear regression performs close predictions.

C. Predictors comparison

Regarding the accuracy of the Analytical Predictor with respect to the Self-Adjusting Predictor, we show

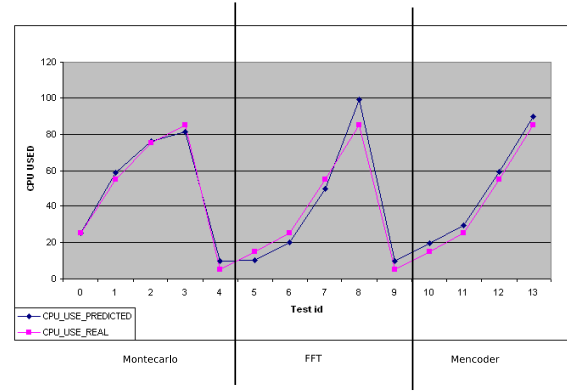


Fig. 5. Comparison between the real CPU usage and the predicted one using the prediction algorithm M5.

TABLE I
SIMPLE LINEAR REGRESSION ACCURACY.

Application	Correlation coefficient	Mean absolute error
FFT	0.9996	0.0003
Montecarlo	0.9998	0.0003
Mencoder	0.9991	0

the accuracy for the same set of predictions in Table II.

We can see that the Analytical Predictor is able to make a prediction with an average absolute error of 1.57 unit of percentage of CPU in the worst case, in contrast with the Linear Regression which achieves an average absolute error of the 0.0003. Nevertheless we consider acceptable the Analytical results while the Self-Adjusting Predictor module is being trained.

TABLE II
ANALYTICAL PREDICTOR (AP) AND LINEAR REGRESSION (LR) ACCURACY COMPARISON.

Application	Mean absolute error (AP)	Mean absolute error (LR)
FFT	1.573067	0.0003
Montecarlo	0.918	0.0003
Mencoder	0.815	0

VII. RELATED WORK

In the literature, as far as we know, there are no articles about predicting the required amount of CPU to execute a job. However, there are a lot of work done around the execution time prediction. We divided this work whether it requires user information, as we do, to work properly or not. Within the first group we find [4] [5] which predict the execution time using time series analysis and taking into account the CPU load of the system. They made the same assumption that us, which is providing a good approximation of the task execution time. Our work

differs from theirs in the sense that we only need this information during a warm up phase, while the Self-Adapting Predictor is learning. Moreover, our system is able to work properly even if a bad approximation was provided. On the other hand, Smith et al. [6] propose a prediction mechanism using historical information from similar tasks already executed by the system. To determine whether or not two tasks are similar they define a set of templates based on job characteristics (e.g. (user,executable)). They consider several prediction techniques such as mean, linear regression, inverse regression, and logarithmic regression. Instance-based learning (a.k.a. locally weighted learning techniques) is used in [7] to construct a database of experiences where each experience consist of input and output features. Input features include who submitted the job, the executable, the number of CPUs requested etc. and output is represented by the execution time. To determine how relevant each data of the database is to predict the execution time, they use a specific distance function namely *Heterogeneous Euclidean Overlap Metric*. Then, they predict the execution time using a Gaussian function to form a distance-weighted average.

Finally, Zhang et al. [9] present another work in line with ours, which consists in a framework for anomaly detection and capacity planning for multi-tier services that uses a regression-based solver to derive the CPU demand of each web transaction type on a given hardware.

VIII. CONCLUSIONS AND FUTURE WORK

We end this paper presenting our conclusions and outcomes. We have contributed with an architecture for resource providers that handle heterogeneous workloads in virtualized resources. We also have implemented a dual-purpose predictor to help providers to know in advance if they will meet jobs dealines and decide whether to accept or reject the job at submission time. Besides, they also can discard a job during its life time in order to accomodate a more prioritized job. On the other hand, they are able to do an efficient allocation of resources to jobs. It is so because the prediction provides the minimum amount of CPU required for a job to meet its deadline.

Regarding our future work it includes:

- Introduce other resources (e.g. memory, bandwidth) in the resource requirements prediction. This includes deling with not only CPU-intensive jobs.
- Determine automatically when the Scheduler module should query the Analytical Predictor module or the Self-Adjusting Predictor module.
- Include the prediction of the requirements for the web workload in the Prediction System.

ACKNOWLEDGEMENT

This work is supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2007-60625 and

by the Univeristat Politècnica de Catalunya (UPC) under grant UPC-RECERCA.

Thanks to The Grid Workloads Archive [8] and the Grid'5000 team for the workload traces.

REFERENCES

- [1] Moorsel A.V. *Metrics for the Internet Age: Quality of Experience and Quality of Business*. HP Labs Technical Report HPL-2001-179, July 2001. Fifth International Workshop on Performability Modeling of Computer and Communication Systems, Arbeitsberichte des Instituts für Informatik, Universität Erlangen-Nürnberg, Germany, Band 34, Nummer 13, pp 26-31, September 2001
- [2] Wang, I and Witten, I. H. *Induction of model trees for predicting continuous classes* Poster papers of the 9th European Conference on Machine Learning, Springer, 1997
- [3] Tseng, S.M. Chin, Y.H. Yang, W.P. *Scheduling real time transactions with dynamic values: a performance evaluation* Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications, Tokyo, Japan, October 1995, pp. 60-67
- [4] Padgett, J. and Djemame, K. and Dew, P. *Predictive Runtime Adaptation for Service Level Agreements on the Grid* 21 stUK Performance Engineering Workshop, 2005.
- [5] Dinda, P.A. *Online Prediction of the Running Rime of Tasks*, Cluster Computing, 2002, volume 5, pages 225236
- [6] Smith, W. and Foster, I. and Taylor, V. *Predicting Application Run Times Using Historical Information*, Lecture Notes in Computer Science, 1998, volume 1459, pages 183
- [7] Smith, W. *Improving Resource Selection and Scheduling Using Predictions*, Kluwer Academic Publishers Norwell, MA, USA, 2004
- [8] *The Grid Workloads Archive*.
<http://gwa.ewi.tudelft.nl/pmwiki/>
- [9] Zhang, Q. and Cherkasova, L. and Smirni, E. *A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications* Proc. of the Fourth International Conference on Autonomic Computing (ICAC'2007), Jacksonville, FL