Prediction of Job Resource Requirements for Deadline Schedulers to Manage High-Level SLAs on the Cloud

Gemma Reig, Javier Alonso, and Jordi Guitart
Universitat Politècnica de Catalunya (UPC) and Barcelona Supercomputing Center (BSC)
Barcelona, Spain
Email: {greig, alonso, jguitart}@ac.upc.edu

***Abstract*—For a non IT expert to use services in the Cloud is more natural to negotiate the QoS with the provider in terms of service-level metrics –e.g. job deadlines– instead of resource-level metrics –e.g. CPU MHz. However, current infrastructures only support resource-level metrics –e.g. CPU share and memory allocation– and there is not a well-known mechanism to translate from service-level metrics to resource-level metrics. Moreover, the lack of precise information regarding the requirements of the services leads to an inefficient resource allocation –usually, providers allocate whole resources to prevent SLA violations. According to this, we propose a novel mechanism to overcome this translation problem using an online prediction system which includes a fast analytical predictor and an adaptive machine learning based predictor. We also show how a deadline scheduler could use these predictions to help providers to make the most of their resources. Our evaluation shows: i) that fast algorithms are able to make predictions with an 11% and 17% of relative error for the CPU and memory respectively; ii) the potential of using accurate predictions in the scheduling compared to simple yet well-known schedulers.**

I. Introduction

Recently, the Cloud has suffered a huge expansion opening the IT complex infrastructures to users with quite different IT background. For these users to use the Cloud they are required to negotiate a Service Level Agreement (SLA) with the Cloud service provider specifying the conditions of the service. Usually, these users know, for example, when they want their submitted tasks to be completed. However, they tend to do not know how many resources have to buy to achieve their target. Thus, for a non IT expert to use services is more natural to negotiate the SLA with the provider in high-level terms, representing the quality of their experience [1]. That is, expressing the QoS with service-level metrics –e.g. job deadlines– instead of resource-level metrics –e.g. CPU MHz–, which could depend on the underlying architecture and thus, they are not transferable from a Cloud provider to another. On the other hand, dealing with service-level metrics is also an advantage for the provider as it has more freedom deciding the resource allotments to users' applications whilst it fulfills their SLAs.

Moreover, the Cloud accommodates applications that exhibit very heterogeneous behavior. This heterogeneity has different dimensions: the nature of the application –ranging from HPC jobs to Web services–, the application's use of resources –e.g. CPU-bounded– and the workload as the arrival of jobs is variable. In order to be profitable, service providers tend to share their resources, by means of virtualization, among multiple concurrent applications owned by different users. However, because of applications heterogeneity, it is not assumable to statically provision dedicated resources to the applications, since this leads to an inefficient resource utilization. The solution is dynamically allocating resources to the applications depending on their needs, while guaranteeing that each of them has always enough resources to meet the agreed service-level metrics. In such an scenario, if the provider does not have precise information regarding the requirements of the services, this leads to an inefficient resource allocation –usually, providers allocate whole resources to prevent SLA violations.

For this reason, although service-level metrics are preferred by both users and providers, the latter require resource-level information in order to allocate the adequate amount of resources to the applications and charge for their execution. Thus, the provider should translate somehow these service-level metrics to resource requirements. However, there is not a well-known mechanism to translate from service-level metrics to resource-level metrics efficiently. That is, to decide the resource requirements to fulfill the service-level metrics.

Our contribution targets virtualized software as a service (SaaS) providers that handle heterogeneous workloads. These providers dedicate part of their resources to execute web applications, and try to accommodate batch jobs in the remaining resources. We propose a prediction system to determine the minimum job resource requirements to be executed before its deadline. One key innovation of the prediction system is the usage of Machine Learning (ML) to enable the translation from service-level metrics to resource requirements. In addition, we demonstrate how different scheduling policies can take advantage of these predictions to do a more efficient resource allocation. In particular, we devise the architecture of a scheduler that enables the scheduling policies to discard the execution of a job in advance if they predict that it will not finalize before its deadline, according to the resources available in the system in a specific moment.

Thus, our contribution's aim is twofold: i) enabling the Cloud to non-expert IT users by means of using service-level metrics and ii) help providers to do a smart utilization of their resources by using the resources left by web applications to

execute jobs in an efficient way –e.g. discard jobs in advance, avoiding the risk of wasting resources in executing jobs that will not meet their deadlines. An extended version of this paper is presented in [2].

The rest of this paper is structured as follows: Section II presents our assumptions, Section III describes our proposal's architecture, Sections IV and V present the scheduler and the prediction system respectively, Section VI shows the results obtained, Section VII presents the related work and finally, Section VIII exposes our conclusions and future work.

II. Environment and assumptions

For the scope of this paper, we make some assumptions: i) although we envision a heterogeneous workload, this work focus on providing resource predictions for repetitive CPU and memory intensive single threaded batch jobs. We do not deal with I/O intensive jobs because the virtualization mechanism used in the experimentation does not provide an interface to manage I/O allocation among VMs [3]. Predicting the resource requirements for the web workload is part of our future work. ii) SLAs for batch jobs have to specify the job to execute and the deadline date. Also, this SLA should reflect rewards and penalties to be applied when the job's deadline is not met. We consider the approach in which the penalties do not exceed the reward to avoid malicious clients who would rather prefer to receive indemnities than to execute jobs.

III. Architecture

This section presents the logical architecture of the system which is composed by two main components:

- The **Scheduler** accepts incoming jobs and web applications to be planned. It queries the *Prediction System* and it decides, depending on the policy being used and the resources' status, how to allocate resources to the incoming jobs and how to elastically size up and down the resource allocation for web applications, in order to fulfill their respective QoS.
- The **Prediction System** is in charge of predicting the minimum resource requirements needed to meet SLAs. It consists of an *Analytical Predictor (AP)* module and a *Self-Adjusting Predictor (SAP)* module that predicts by learning from previous job executions.

IV. Scheduler

This section describes the *Scheduler*, which uses the prediction mechanisms to make the most of provider's resources. Once a job arrives at the system, the *Scheduler* queries the *Prediction System* about the minimum resource requirements of the job to meet its deadline. The *Scheduler* receives a prediction from both the *AP* and the *SAP*. Initially, it only takes into account the prediction made by the *AP*, as the *SAP* is not trained yet. Once the *SAP* is trained and therefore, it performs lower error than the other –as it is shown in the experimentation Section VI– the *Scheduler* starts to obey the *SAP*.

Using this prediction and the resource status information, the *Scheduler* might decide (depending on which scheduling policy it is being used): i) to discard the job if the amount of resources predicted is not available and consequently the job cannot be executed on time; ii) to run the job allocating to it the required resources; iii) to delay this decision along the lifetime of the job –e.g. the provider may accept a job, but later on discard it if a new job with highest priority arrives at the system and both SLAs cannot be fulfilled.

The *Scheduler* obtains from the *Prediction System* the minimum amount of required resources to execute a job before its deadline. However, if there are more resources available, the *Scheduler* allocates them to the job. This way, jobs potentially make more progress than the strictly required to meet the deadline. For example, CPU-bounded jobs will make more progress as more CPU they have allocated. However, a greater allotment of memory does not imply a decrease in the execution time –if the task working set already fits in memory, increasing the allocated memory to the task will not decrease its execution time. Thus, the provider gets ahead with the work, by means of fully using the resources when the system is underloaded in order to have more resources available for future overload periods. Thanks to virtualization, changing dynamically the resource allocation is straightforward [3].

Once a job is completed, the *Scheduler* stores the job's execution information –job, resource allotment and execution time– in the *Prediction System* to be used to train the *SAP*.

V. Prediction System

The *Prediction System* stores and manages information of previous job executions in order to predict the amount of required resources –CPU share and memory allocation– by a job to be completed before its deadline. This prediction is carried out by means of two components: the *AP* and the *SAP*.

A. Analytical Predictor (AP)

The aim of the *AP* is to predict job resource requirements while the *SAP* is not trained.

Regarding the prediction of CPU requirements, we approximate analytically T_{cpu}, which is the execution time of a CPU-bounded job assuming that it has allocated a CPU share of $\%CPU$, by the following equation: $T_{cpu} = \frac{T_{ref}}{\%CPU}$, where T_{ref} is a reference of the execution time of the job weighted by the CPU share that was allocated to that execution e ($T_{ref} = T_{cpu,e}\%CPU_e$). This data is obtained from the most similar –in terms of execution time– past execution of the same application with the same input data size.

Thus, the *AP* module can determine the minimum CPU requirements as follows: $\%CPU = \frac{T_{ref}}{Deadline - Now}$, where $Deadline$ is the deadline for executing the job, and Now is the current time. According to this, whenever the Deadline is in the future ($Deadline > Now$), $\%CPU$ is a positive number. Then, the deadline potentially can be accomplished if $\%CPU \leq 100$.

Whereas the amount of CPU allocated to a job directly determines its execution time, there are some differences when we are considering memory. In particular, whilst the amount of memory allocated to a job is greater or equal to the amount

of memory it requires, the execution time is always the same, even if we allocate more memory to the job. Otherwise, if the memory allocated is lower than the required one, the job starts to use the swapping space and the lesser is the memory allocated to the job the longer is its execution time. Besides, there is a high variability of the execution time when the memory allocated is lower than the job requirement, making the analytical prediction imprecise. However, we use the same idea discussed for the CPU to predict analytically the memory requirement because it might be useful while the *SAP* is not trained.

Thus, the *AP* is useful to make predictions for CPU or memory bounded jobs whilst the *SAP* is not trained. Nevertheless, it is not able to predict the requirements of a job that uses both memory and CPU.

B. Self-Adjusting Predictor (SAP)

The *Self-Adjusting Predictor* uses ML based models to predict the CPU share and the amount of memory required to execute a job before its deadline expires. The *AP* is non-flexible to changes in the behavior of the jobs and it requires a past execution of the same job with the same input data size to predict the resources needed. But the *SAP*, thanks to the use of ML, offers a more flexible behavior, allowing to predict the resources needed by an application, without needing any previous execution of the same job. Furthermore, the *SAP* has the capability to learn from different jobs and predict the resources needed from completely different jobs using a single model. Furthermore, new job executions are continuously used to build new models, which replace the used ones when higher accuracy is achieved.

We have chosen some ML algorithms to build the models that perform the predictions: Linear Regression, M5P [4], REPTree and Bagging [5]. Basically because their prediction accuracy is high and their computational complexity is low –introducing a low overhead, which is important in an on-line system. Through the experimentation in Section VI, we evaluate which algorithm is more suitable to predict each resource.

VI. Experiments

To evaluate our proposal, we first evaluate the accuracy of the *SAP*. Then, we compare the two prediction modules and estimate the learning curves for the *SAP*. Finally, we show that the *Prediction System* is useful for discarding unfeasible jobs.

A. Experimental environment

We test our proposal by means of simulation. First, the simulation of the *SAP* uses the implementation of the ML algorithms provided by the WEKA toolkit [5] to build the prediction models. In order to train and test these models, we have used Euler, MolDyn, RayTracer, and Montecarlo benchmarks from the Java Grande Benchmarks [6] and the Mencoder application as CPU-bounded applications. Also, a microbenchmark that allocates 1924 MB of memory and access it randomly as a memory-bounded application, and finally, another microbenchmark as a CPU and memory intensive application with the following structure: its initialization phase uses CPU and memory; the second phase is intensive in memory; and finalizes with a CPU intensive phase. These applications run in a Xen virtual machine with an Intel(R) Xeon(TM) MP CPU 3.16GHz. In order to evaluate the performance of the built models, we have used two different datasets: one for training and one, totally different to validate the model.

Second, to evaluate the usefulness of the predictions for scheduling jobs, we have developed a *discrete time* simulator in Java, which uses the *AP* to perform its predictions. The input workload of the simulator uses the workload of Grid'5000 [7], completing it with the additional job information we need. Specifically, it randomly sets the revenue for each job in the range $[0, 1)$ and generates a workload web assuming a sinus-periodic occupation of the resources between 0 and 100% (to simulate peak and off-peak hours). We also set to 20% the minimum CPU share that can be allocated to a job.

B. Self-Adjusting Predictor accuracy

In this section, we show the accuracy of the predictions performed using the ML algorithms mentioned in Section V-B. We test these algorithms in four different scenarios: i) to predict the amount of CPU for a job; ii) to predict the amount of CPU for a job taking into account the size of its input data; iii) to predict the amount of memory allocated to a job; iv) to predict both CPU or memory.

1) Predicting CPU: In Section V-A, we have explained that the relation between the CPU allocated to a job and its execution time varies as $\frac{T_{ref}}{T_{cpu}}$. Thus, if we transform the examples into sets of CPU allocation, the inverse of the execution time and the application's name, we would have a linear relation between the CPU share and the execution time. With this transformation we help the models to deduce the relation between CPU and execution time. Thus, we train the aforementioned ML algorithms with these transformed examples.

The accuracy of these models in terms of mean absolute errors (MAE) is shown in Table I. M5P is the algorithm that performs better predictions on average –2.35 units of CPU percentage in absolute terms, or 5% of relative error– followed by Bagging with M5P.

TABLE I
Mean absolute error (units of %CPU) predicting CPU.

LR	M5P	REPTree	Bagging M5P	Bagging REPTree
10.78	2.35	5.04	3.04	4.18

We also have studied the possibility of predicting each application separately instead of using a single model for all of them. That is, train the algorithms first with the examples of Montecarlo executions to build a model for the Montecarlo application and so forth for each application. Although having one application per model leads to improved accuracy we prefer to build a single model to avoid handling with unlimited models as the amount of applications might grow. Also, this way we could try to predict unseen applications using the information learned with the ones we know.

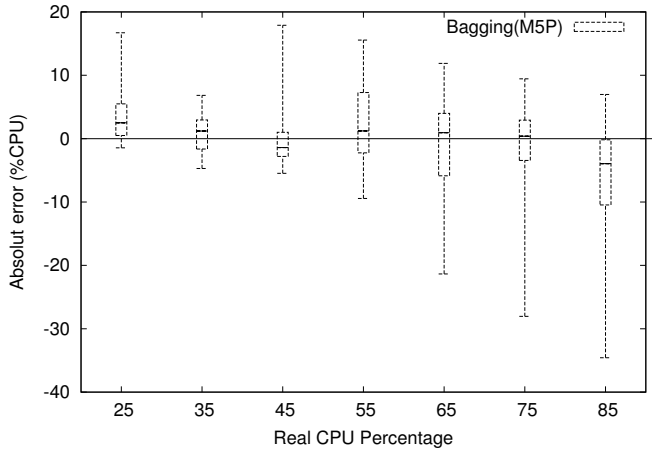

Fig. 1. CPU prediction accuracy taking into account input data size.

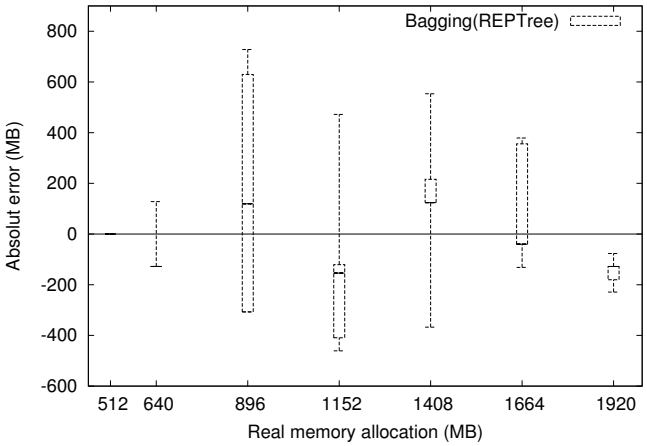

Fig. 2. Memory prediction accuracy evaluation

2) Predicting CPU regarding input size: In this section, we consider the same problem but now considering that the applications have different input data sizes. For the Mencoder and the Java Grande applications, we consider 5 sizes for training and 4 sizes for testing purposes. In this case, we also have applied a transformation to the examples. It consists of deriving a new attribute that represents that CPU requirements are proportional to the size of the input data and inversely proportional to the execution time ($\frac{input\_data\_size}{execution\_time}$). Table II shows the MAEs of the models built using each of the ML algorithms. We can observe that Bagging with M5P is the best algorithm to make this kind of predictions –11% of relative error. Figure1 shows the distribution of the differences between the real CPU percentage required to meet the deadline with the predicted CPU percentage for the Bagging with M5P model. It shows 5, 25, 50, 75 and 95 percentiles. The horizontal line in the figure separates those predictions that surpass the strictly required CPU percentage from those where the predicted CPU is lower than the required one. We make this distinction because the second case is more problematic for the provider as it leads to a deadline omission violating the SLA. As opposite, if the predicted amount of CPU is greater than the required one, we are over-provisioning the application with resources, though the job would probably use this CPU surplus to finish its execution earlier. In this case, the greater the amount of CPU to predict the worst predictions. The knowledge of the error tendency could help us in designing scheduling policies that deal with these errors.

TABLE II
MEAN ABSOLUTE ERROR (UNITS OF %CPU) PREDICTING CPU TAKING INTO ACCOUNT INPUT DATA SIZE.

Application	LR	M5P	REPTree	Bagging M5P	Bagging REPTree
all	15.39	8.69	8.92	6.77	7.11

3) Predicting memory: To study how well the amount of memory required by an application may be predicted, we use the same applications as in the case of CPU plus a memory intensive microbenchmark. We train and test the models with real executions using several memory allotments. The behavior of the applications is the following: if allocated with the required memory or greater the execution time does not vary –only varies lightly if the application is implemented in Java because the garbage collector is executed more often with lower allotments– and thus, the memory allotment and the execution time are not correlated, making impossible to predict accurately. For this reason, we preprocess automatically the dataset changing the amount of memory of a sample if there is a similar sample –in terms of execution time– with less memory. As opposite, when the memory allotment is lower than the required the application begins to use the swap area increasing the variability of executions with the same allocation and thus, reducing the prediction accuracy with respect to the CPU case, as shown in Table III, which shows how accurate are the predictions made by the algorithms selected, and Figure 2, which shows the error distribution for the more accurate algorithm –i.e. Bagging with REPTree with 17% of relative error. As a result of the preprocess, there are not the same number of examples for each allotment. For instance, 512MB has 17% of the examples as most of the examples corresponding to CPU intensive applications have collapsed in the minimum memory allotment. As opposite, 1920MB only has 4% of the examples –see pondered results in Table III.

TABLE III
MEAN ABSOLUTE ERROR (MB) PREDICTING MEMORY.

Application	LR	M5P	REPTree	Bagging M5P	Bagging REPTree
all	190.67	200.45	178.06	179.45	158.35

4) Predicting CPU and memory: Although we can combine both resources in a single model, we can only predict a resource at once –because there are several possible combinations of CPU and memory allotments for a job to be finished

before its deadline. In order to overcome this difficulty, we offer different solutions: one possibility is to predict the execution time of the job assuming that it has allocated all the available resources. This way, if the predicted time is lower than the time remaining to the deadline, the provider could accept the job. A similar solution consists of fixing the amount of one resource and predicting the other. And the last proposal is to combine two predictions, but we discourage this approach as its total error accumulates the error of the two individual predictions.

Table IV shows the accuracy of the models predicting each resource. These models are trained and tested with all the applications used before to test memory and CPU separately, plus a CPU and memory intensive microbenchmark. The model with highest accuracy to predict CPU is Bagging with M5P with a 14% of relative error. Predicting memory is less accurate –for the reasons explained in Section VI-B3– with a relative error ranging from 36 to 43%. Finally, to predict execution time Bagging with M5P has a 15% of relative error.

TABLE IV
MEAN ABSOLUTE ERROR PREDICTING CPU AND MEMORY.

Prediction	LR	M5P	REPTree	Bagging M5P	Bagging REPTree
%CPU	18.55	9.20	10.69	7.35	9.66
MEM MB	542.56	508.50	542.60	500.89	506.97
TIME sec	29.43	5.10	10.46	3.97	8.81

C. Predictors comparison

Regarding the accuracy of the *AP* with respect to the *SAP*, we use the same set of data used in Section VI-B1 to train and test both predictors. We have obtained that the *AP* is able to make a prediction with an absolute error of 6.26 units of CPU percentage on average. In contrast, with the *SAP*, when using Linear Regression, the average absolute error is much lesser, 1.62 units of CPU percentage. Nevertheless, we consider acceptable the *AP* results while the *SAP* module is being trained.

This comparison has assumed one model per application and the input data size fixed. The rest of scenarios are not comparable because the *AP* cannot take into account data input sizes or derive a single model for all the applications. The *AP* deals each data input size as a different application. Thus, its usage is only useful while the *SAP* is not enough trained.

D. Learning curves

Although a provider has the *AP* to make predictions whilst the *SAP* is learning, a main concern is the learning speed of the built models.

Figure 3 shows MAE as a function of the number of examples used to build the models to predict the amount of CPU required by a job taking into account the input data size. It shows that with about 500 examples the MAE of the models are near to the results obtained with a larger amount of examples. Similar, in the case of memory only 100 examples are required. Thus, the provider requires few examples to make predictions, which is a desirable characteristic for the system.

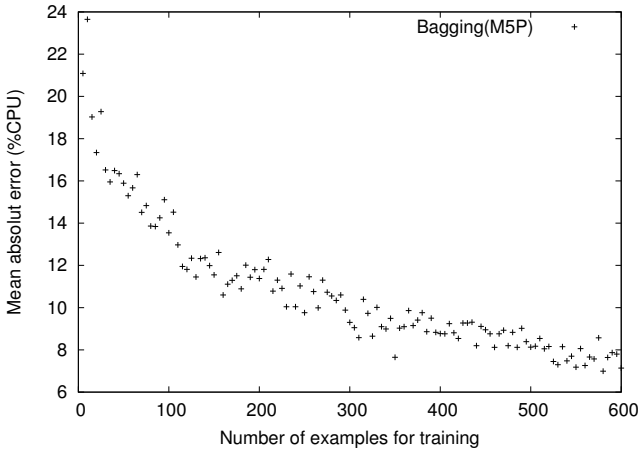

Fig. 3. Learning curve for CPU prediction.

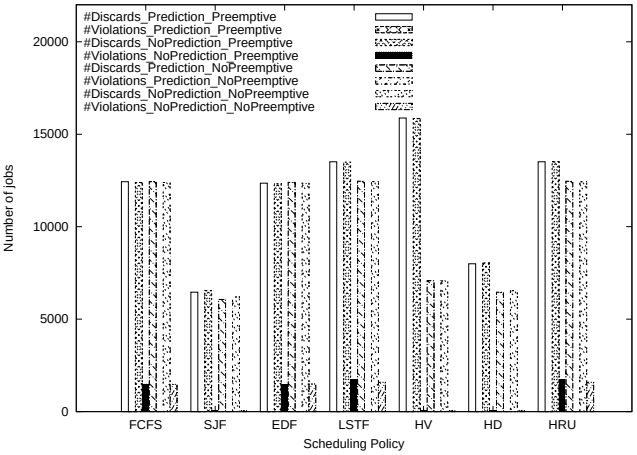

Fig. 4. Jobs discarded and SLA violations with and without using prediction. Policies used: FCFS (First Come First Serve), SJF (Shortest Job First), EDF (Earliest Deadline First) , LSTF (Least Slack Time First), HV (Highest Value), HD (Highest Density), HRU (Highest Reward and Urgency) [8]

E. Prediction usefulness for scheduling

The aim of this test is to demonstrate empirically that having predictions on required resources can help schedulers to efficiently allocate resources to jobs whatever the policy used is. For demonstration purposes, we assume full accuracy on the predictions. According to this, we configure our simulator to simulate 10 CPUs and to run 25000 jobs.

Figure 4 shows the number of jobs being discarded and the number of jobs being violated 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, leading to a potentially waste of resources –the job might be executed partially.

VII. Related Work

Predictive systems which consider a fine-grain management include [9], which predicts CPU and memory requirements to maintain the response time for web servers, and [10], which search for a suboptimal solution in a finite space of possible allocations thus, they do not consider all the possible solutions.

The most similar system to our proposal is Gridbus [11]. It uses a user provided estimation of the job execution time to calculate the number of nodes required for a job to be completed before its deadline. To compute this, the estimation plus the time of staging is divided by the time remaining to the deadline and multiplied by an aggression factor to denote that the system preference is to execute the job as quick as possible. Our system also predicts the requirements to not violate jobs' deadlines. However, our granularity is finest –CPU share and memory allocation– instead of whole nodes and we do not need any user's estimation of the job's execution time.

Many systems instead of predicting the amount of resources focus on predicting the job's execution time. On the one hand, [12], [13] require that the user provides a good approximation of the execution time, then based on the prior CPU load, they predict the job execution time. On the other hand, there are several techniques that do not need any user information in order to perform the execution time prediction, as occurs in [14], which uses historical information from similar tasks – same user, same executable, etc.– to perform the prediction using means and regressions. A locally-weighted learning technique is used in [15] to predict the execution time by means of constructing a database of experiences that include who submitted the job, the number of CPU requested, etc. and which data is the most relevant to perform predictions using a distance function. Our solution, also does not need user's information to perform predictions and our Scheduler exploits the elasticity of virtual machines by using the predicted information to make the most of provider's resources.

VIII. Conclusions and Future Work

In this paper, we have contributed with an architecture to help providers to deal with the heterogeneity naturally present in the Cloud, making the most of their virtualized resources. This architecture includes a dual-purpose predictor that i) allows users to negotiate with providers in service-level terms and ii) provides a mean for the *Scheduler* to perform smart resource allocation using these predictions

We have introduced ML techniques in a *Self-Adjusting Predictor* that predicts the required resources to fulfill a given service-level metric using the results from previous executions. Regarding the CPU prediction, we achieve high prediction accuracy (11% of relative error) using the Bagging with M5P algorithm. Regarding memory prediction, a 17% of relative error is achieved using Bagging with REPTree. Moreover, the learning curves of the built models achieve high accuracy with few examples. This characteristic, in conjunction with the low overhead incurred, makes our system suitable for making decisions online. Besides, we have proposed an *Analytical Predictor* that is used to predict the resource requirements whilst the *Self-Adjusting Predictor* is not enough trained.

Regarding our future work, we are going to build a real prototype of the system by means of integrating the *Prediction System* and the EMOTIVE (Elastic Management Of Tasks In Virtualized Environments) Cloud middleware [16]. On a second place, our research is going to be focused on the resource prediction for web applications and how to predict the requirements of jobs' phases. Finally, we are going to focus on new scheduling policies aware of the predictions.

Acknowledgment

This work is supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2007-60625, by the Generalitat de Catalunya (2009-SGR-980) and by the Univeristat Politècnica de Catalunya (UPC) under grant UPC-RECERCA.

References

[1] A. Van Moorsel, "Metrics for the Internet Age: Quality of Experience and Quality of Business," in *Fifth International Workshop on Performability Modeling of Computer and Communication Systems, Arbeitsberichte des Instituts für Informatik, Universität Erlangen-Nürnberg, Germany*, vol. 34, no. 13, September 2001, pp. 26–31.

[2] G. Reig, J. Alonso, and J. Guitart, "Deadline Constrained Prediction of Job Resource Requirements to Manage High-Level SLAs for SaaS Cloud Providers," Dept. d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, Barcelona, Spain, Tech. Rep. UPC-DAC-RR-2010-9, April 2010.

[3] (2009) Xen api. [Online]. Available: http://wiki.xensource.com/xenwiki/XenApi

[4] Y. Wang and I. Witten, "Induction of Model Trees for Predicting Continuous Classes," *Technical report 96/23. Hamilton, New Zealand: University of Waikato, Department of Computer Science*, 1996.

[5] H. Witten Ian and F. Eibe, "Data Mining: Practical Machine Learning Tools and Techniques," *Morgan Kaufmann, San Francisco*, 2005.

[6] (2009) The java grande forum benchmark suite. [Online]. Available: http://www2.epcc.ed.ac.uk/computing/research_activities/java_grande/

[7] (2009) The grid workloads archive. [Online]. Available: http://gwa.ewi.tudelft.nl/pmwiki/

[8] S. Tseng, Y. Chin, and W. Yang, "Scheduling Real-time Transactions with Dynamic Values: a Performance Evaluation," in *2nd International Workshop on Real-Time Computing Systems and Applications. Tokyo, Japan*, October 1995, pp. 60–67.

[9] Y. Chen, S. Iyer, X. Liu, D. Milojicic, and A. Sahai, "SLA Decomposition: Translating Service Level Objectives to System Level Thresholds," in *4th IEEE International Conference on Autonomic Computing. Jacksonville, Florida, USA*, June 2007, p. 3.

[10] D. Menasce and M. Bennani, "Autonomic Virtualized Environments," in *International Conference on Autonomic and Autonomous Systems, 2006. ICAS'06. Silicon Valley, California*, vol. 6, July 2006, pp. 28–28.

[11] S. Venugopal, X. Chu, and R. Buyya, "A Negotiation Mechanism for Advance Resource Reservations using the Alternate Offers Protocol," in *16th International Workshop on Quality of Service, 2008. IWQoS 2008. University of Twente, Enschede, The Netherlands*, June 2008, pp. 40–49.

[12] J. Padgett, K. Djemame, and P. Dew, "Predictive Run-time Adaptation for Service Level Agreements on the Grid," in *21st UK Performance Engineering Workshop. Nottingham*, July 2005.

[13] P. Dinda, "Online Prediction of the Running Time of Tasks," *Cluster Computing*, vol. 5, no. 3, pp. 225–236, 2002.

[14] W. Smith, I. Foster, and V. Taylor, "Predicting Application Run Times Using Historical Information," *Lecture Notes in Computer Science*, vol. 1459, no. 122, p. 183, 1998.

[15] W. Smith, "Improving Resource Selection and Scheduling Using Predictions," *Grid resource management: state of the art and future trends, Norwell, MA, USA*, pp. 237–253, 2004.

[16] (2009) Elastic Management of Tasks in Virtualized Environments (EMOTIVE). [Online]. Available: http://www.emotivecloud.net/