Resource-Level QoS Metric for CPU-Based Guarantees in Cloud Providers

Íñigo Goiri, Ferran Julià, J. Oriol Fitó, Mario Macías, and Jordi Guitart

Barcelona Supercomputing Center and Universitat Politecnica de Catalunya
Jordi Girona 31, 08034 Barcelona, Spain
{igoiri,fjulia,fito,mario,jguitart}@ac.upc.edu

Abstract. Success of Cloud computing requires that both customers and providers can be confident that signed Service Level Agreements (SLA) are supporting their respective business activities to their best extent. Currently used SLAs fail in providing such confidence, especially when providers outsource resources to other providers. These resource providers typically support very simple metrics, or metrics that hinder an efficient exploitation of their resources.

In this paper, we propose a resource-level metric for specifying fine-grain guarantees on CPU performance. This metric allows resource providers to allocate dynamically their resources among the running services depending on their demand. This is accomplished by incorporating the customer's CPU usage in the metric definition, but avoiding fake SLA violations when the customer's task does not use all its allocated resources. As demonstrated in our evaluation, which has been conducted in a virtualized provider where we have implemented the needed infrastructure for using our metric, our solution presents fewer SLA violations than other CPU-related metrics.

Keywords: QoS metrics, SLA, Cloud provider.

1 Introduction

The emergence of Cloud computing solutions has attracted many potential consumers, such as enterprises, looking for a way to reduce the costs associated with supporting their business processes. Using the Cloud, these customers can outsource services (offered by service providers) which can be easily composed to build distributed applications. Additionally, the Cloud allows resource providers to offer raw resources as a service where the consumers can outsource the execution of their own services. In the same way, service providers can also use the Cloud to overcome an unexpected demand on their services by outsourcing additional resources to other providers, or even they can fully rely on external resources to provide their services. In all these scenarios, it is highly desirable that the consumers receive fine-grain Quality of Service (QoS) guarantees from the providers. However, this becomes essential when service providers outsource

J. Altmann and O.F. Rana (Eds.): GECON 2010, LNCS 6296, pp. 34–47, 2010.

resources to resource providers, since the QoS guarantees they can provide to their respective customers depend on the QoS guarantees they receive from the resource providers.

Typically, a provider agrees the QoS with its customers through a Service Level Agreement (SLA), which is a bilateral contract between the customer and the provider (or between providers) that states not only the conditions of service, but also characterizes the agreed QoS between them using a set of metrics. Service providers naturally offer service-level metrics (e.g. service execution deadline [6]) to their customers for specifying the QoS. Using service-level metrics has advantages for both the customer and the provider. The former does not need to provide detailed information about the resource requirements of the service to be executed (probably the customer does not know this exactly), but only a high-level performance goal (e.g. a deadline for executing the service). The latter can freely decide the allocated resources to the service whereas it guarantees that the service meets the agreed performance goals. Being able to dynamically allocate resources to the different services is especially important for Cloud providers considering that, aiming for profitability, they tend to share their resources among multiple concurrent services owned by different customers.

On the other hand, resource providers in the Cloud must offer resource-level metrics that can be used to provide fine-grain QoS guarantees. First, the QoS agreement can be naturally expressed using resource-level metrics (e.g. number of processors, frequency of processors, FLOPS, etc.), since raw resources are the traded good. Second, having fine-grain metrics, which guarantee a given resource allocation during a time period, is especially important for service providers that outsource resources to resource providers, as we have stated before. For instance, try to figure out how a service provider could guarantee that a service will finish within a given deadline if he does not receive fine-grain guarantees on the FLOPS supplied at every instant by the resource provider where the service is running.

Furthermore, some service providers can also benefit in some situations from supporting resource-level metrics. First, those that do not support the inference of the resource requirements of a service to fulfill a given service-level metric, and for this reason, cannot offer service-level metrics in the SLA. Second, those dealing with customers (typically coming from the HPC domain) that prefer to stay with the resource-level metrics that they have been using for a long time, instead of moving to service-level metrics.

Nevertheless, current Cloud providers do not support fine-grain resource-level QoS guarantees on their SLAs. In fact, most of them only support SLAs with very simple metrics based on resource availability [2,12,14]. For instance, whereas Amazon EC2 [4] offers different instances according to their computing capacity (which is measured in ECUs, EC2 compute units), there is not any guarantee in the SLA that this computing capacity will be supplied during the whole execution of the instance, as Amazon's SLAs only provide availability guarantees [12].

According to this, one could think on using Amazon's ECUs to support fine-grain resource-level guarantees on Cloud SLAs, considering also the porting of traditional resource-level metrics from the Grid environment to Cloud providers.

However, this must be carried out carefully, since it can prevent the providers from obtaining the maximum profit of their resources if done naively. In particular, metrics related to the provider's computing capacity (i.e. CPU) are especially susceptible to naive usage. For instance, if the agreed metric in the SLA establishes that the number of processors allocated to a given service must be greater or equal to some value, the provider must maintain this assignment during the whole execution of the service, even if that service is not using all the allocated capacity during some phases of its execution (i.e. the provider is forced to statically overprovision processors to the service to avoid SLA violations). Notice that the unused resources could be temporarily allocated to another service, improving in this way the utilization and the profit for the provider.

In this paper, we derive a resource-level metric for specifying fine-grain QoS guarantees regarding the computing capacity of a Cloud provider by extending the Amazon's approach. Our metric overcomes the limitations of traditional CPU-related metrics. By taking into account the customer's resource usage, it allows the provider to implement dynamic resource provisioning and allocate to the different services only the amount of CPU they need. In this way, better resource utilization in the provider can be achieved. In addition, it avoids fake SLA violations, which we define as those situations where the SLA evaluator mechanism detects that an SLA is being violated, the provider will be penalized for this, but the violation is not provoked by the provider's resource allocation. In general, this occurs when the provider uses a poorly designed resource-level metric and the customer's service does not use all the resources it has allocated. Of course, the customer is free to use the amount of resources he wants, and this must not provoke any SLA violation. Therefore, the solution is to design solid resource-level metrics that support this. Finally, the proposed metric can be used in heterogeneous environments, since it is based on Amazon's ECUs.

The remainder of this paper is organized as follows: Section 2 explains our approach to deal with platform heterogeneity. Section 3 describes the derivation of our resource-level CPU metric. Section 4 presents the experimental environment where this resource-level metric has been implemented. Section 5 describes the evaluation results. Section 6 presents the related work. Finally, Section 7 presents the conclusions of the paper and the future work.

2 Metric Unification among Heterogeneous Machines

Cloud providers typically present very diverse architectures: different processor models, each of them with different clock frequencies, etc. For this reason, a good resource-level metric has to be platform-independent so it can be used in all these architectures. In this section, we describe our approach for unifying computing capacity metrics among machines with heterogeneous architectures.

Commonly, *megahertz* have been used to measure the computing capacity of a machine. Nevertheless, this does not directly measure the computing power of a machine, since noticeable differences can be observed depending on the processor architecture. For instance, using this measure a Intel Pentium III with 500 MHz would be 20 times slower than a Intel Xeon 4-core with 2.6 GHz ($\frac{4 \cdot 2600}{500} = 20$). However, simple tests demonstrate that it can be up to 85 times slower.

In order to consider the heterogeneity of the different processor architectures, Amazon uses EC2 compute units (ECU) in its services [4]. An ECU is equivalent in CPU power to a 1.0-1.2 GHz 2007-era AMD Opteron or Intel Xeon processor. This serves as a unified measure for the computing power, though it is not easily portable among different architectures. In this work, we use ECUs in order to unify CPU-related SLA metrics among heterogeneous machines, and additionally we extend the Amazon's approach in order to set up SLAs that provide fine-grain CPU guarantees based on ECUs during a time period.

In our proposal, the maximum computing capacity of a given machine is measured using its maximum amount of CPU[1] ($maxCPU$) and the $ECUs$ associated to the processor installed in that machine ($\frac{maxCPU}{100} \cdot ECUs$).

3 Derivation of CPU-Based SLA Metric

This section describes how our resource-level metric for establishing computing power guarantees is derived, and at the same time, discusses the limitations of alternative metrics. All the metrics discussed in this section intent to establish a guarantee on the computing performance (in terms of CPU) of a service over a time period in a given provider using the idea presented in the previous section. This guarantee is a fixed value for each SLA, represented by the SLA_i term in the formulas, which results from the negotiation between the customer and the provider. The customer is only required to specify the computing performance he requires (in terms of ECUs), which will be accepted by the provider if he is able to provide that performance.

The main difference among the metrics is how the amount of CPU for a service is defined. The more natural approach is specifying CPU performance as a function of the allocated CPU to a service, as shown in Equation 1. This metric specifies that the computing power for a service i at every time period t has to be at least the agreed value in the SLA (SLA_i) and depends on the amount of CPU that the provider assigns to the service in that time period ($assig_i(t)$). This assignment can vary over time depending on the provider's status, and it is periodically obtained by means of the monitoring subsystem.

$$\frac{assig_i(t)}{100} \cdot ECUs \geq SLA_i \tag{1}$$

Note that using this metric forces the provider to statically allocate to each customer at least the amount of CPU agreed in the SLA (he can assign more if he wants), because otherwise, the SLA will be violated. Note that there is not any control on whether the customer uses its allocated CPU or not. This is a quite restrictive approach, especially when the customers' services have a variable

[1] All CPU-related measures are quantified using the typical Linux CPU usage metric (i.e. for a computer with 4 CPUs, the maximum amount of CPU will be 400%).

CPU usage over time. In this case, the provider will suffer from low resource utilization when the services do not use all the CPU they have allocated, since unused resources cannot be allocated to other services running in the provider.

As we have commented before, the provider aims to dynamically allocate its resources to the services depending on their demand, in order to improve resource utilization (and then increase profit). This requires an SLA metric that considers the CPU usage of the services. However, as the CPU usage depends on the client's task behavior, it must be carefully used as a CPU guarantee because it can provoke undesired effects.

For instance, Equation 2 shows an SLA metric where the computing power for a service i at every time period t depends on the amount of CPU that the service uses in that time period ($used_i(t)$). This metric assumes a provider that is able to predict the CPU usage of a given service during the next time period using the CPU usage measures of the service from the previous time periods. This could be achieved with reasonable accuracy using techniques such as Exponential Weighted Moving Average (EWMA). The provider will assign CPU to the service according to its CPU usage prediction. This allows the provider to dynamically allocate the CPU among the services whereas it assigns each service at least with the CPU required to fulfill the SLA.

$$\frac{used_i(t)}{100} \cdot ECUs \geq SLA_i \tag{2}$$

When using this metric, an SLA could be violated in two situations. First, when the provider assigns to the service an amount of CPU that is not enough to fulfill the SLA. This is a *real* violation, the provider is responsible for it, and must pay the corresponding penalty. Second, when the provider assigns to the service an amount of CPU that should be enough to fulfill the SLA, but the service does not use all the assigned CPU. This is what we have defined as *fake* violation, since the provider is not causing it, and for this reason, he should not pay any penalty. However, Equation 2, as currently defined, provokes this to be considered as a *real* violation, thus penalizing the provider for it.

Of course, the service should be able to use only a part of the CPU it has assigned without incurring on SLA violations. In order to allow this, we introduce our metric, in which we introduce a factor that represents the percentage of CPU used by the service with respect to its total amount of allocated CPU. As shown in Equation 3, when the service is using all the assigned resources, Equation 1 applies, so an SLA violation will only arise when the assigned resources are not enough to fulfill the SLA. When the service is not using all the allocated resources then the SLA is considered to be fulfilled, since the provider is not responsible that the service does not exploit its allocated resources.

$$\frac{assig_i(t)}{100} \cdot ECUs \geq SLA_i \cdot \left\lfloor \frac{used_i(t)}{assig_i(t)} \right\rfloor \tag{3}$$

However, some services, even being CPU-intensive, do not use the 100% of their assigned CPU during their whole execution. For these services, Equation 3 does not work. In order to overcome this limitation, we introduce an α factor as shown in Equation 4. This factor acts as a threshold to choose when the service is considered to be using all its assigned resources. For instance, if we consider that a service is using all its allocated resources when it reaches a 90% utilization, α should be set to 0.1.

$$\frac{assig_i(t)}{100} \cdot ECUs \geq SLA_i \cdot \left\lfloor \frac{used_i(t)}{assig_i(t)} + \alpha \right\rfloor \tag{4}$$

Operating on Equation 4, we obtain the version of our metric ready to be used in an SLA:

$$\frac{\frac{assig_i(t)}{100} \cdot ECUs}{\left\lfloor \frac{used_i(t)}{assig_i(t)} + \alpha \right\rfloor} \geq SLA_i \tag{5}$$

Notice that equation in this form can have an undefined value when the denominator is zero, which happens when the service is not considered to use all its allocated resources. We avoid this by defining SLA_i' as $1/SLA_i$ and operating the equation. Notice that, when using this metric, the value specified in the SLA is SLA_i' instead of SLA_i. The final version of our metric is as follows:

$$\frac{\left\lfloor \frac{used_i(t)}{assig_i(t)} + \alpha \right\rfloor}{\frac{assig_i(t)}{100} \cdot ECUs} \leq \frac{1}{SLA_i} = SLA_i' \tag{6}$$

4 Experimental Environment

In order to compare how our metric performs with respect to other CPU-based metrics, we use the SLA enforcement framework built on top of the virtualized provider presented in [15]. This provider, which has been developed within the BREIN European IST Project [9], uses virtual machines to execute the tasks, which allows taking advantage of virtualization features such as migration and easy resource management. In addition, virtualization allows the consolidation of services in the same physical resources, which enhances resource utilization. However, if the amount of resources assigned to these services is static and does not consider the real resource usage, the underutilization problem remains, as we have discussed in this paper. The virtualized provider also has a monitoring subsystem that allows easily consulting the amount of CPU allocated to a VM ($assig_i$) and the amount that it is really using ($used_i$). In addition, the ECUs of the underlying machine can be also calculated from the processor model, its frequency, and its associated *BogoMips* [11]. These *BogoMips* are used to convert the computing power among different architectures.

The SLA framework allows assigning its own SLA to each service by using a XML description that combines both WS-Agreement [5] and WSLA [16] specifications. Using these specifications, we can accurately define the metrics derived in the previous sections. In addition, we use some features of the above-mentioned SLA specifications to define the window size of the average (i.e. 10) and the interval between two consecutive measures (i.e. 2 seconds).

Each SLA S_i specifies the revenue that the customer will pay if the SLA is fulfilled ($Rev(S_i)$), and the penalty that the provider will pay otherwise ($Pen(S_i)$). According to this, the provider's profit for running a given application ($Prof(S_i)$) results from the revenue paid by the customer minus the penalties that the provider has to pay due to SLA violations, i.e. $Prof(S_i) = Rev(S_i) - Pen(S_i)$.

In order to establish the penalties, we use a methodology similar to the one presented in [13], which is built over the same Cloud infrastructure. Each penalty $Pen(S_i)$ is calculated as a percentage of the revenue obtained when fulfilling the corresponding SLA in the following way: $Pen(S_i) = Rev(S_i) \cdot \frac{Gom(\sigma(S_i))}{100}$. This percentage is calculated by using a Gompertz function, which is a kind of *sigmoid function*. Its basic form is $y(t) = ae^{be^{ct}}$, where a is the upper asymptote, c is the growth rate, and b, c are negative numbers.

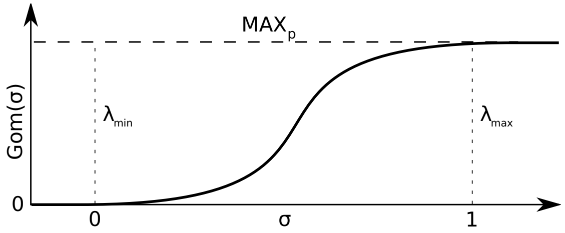

Fig. 1. Gompertz Function for SLA Penalties

For our purposes, we have adapted this function as shown in Figure 1, which displays the penalty percentage depending on a $\sigma(S_i)$ function that represents the SLA violation ratio. In particular, as shown in Equation 7, when this ratio is zero or less, the penalty percentage is 0. When this ratio is one, the percentage tends to $MAX_P\%$ of the price that the client pays for SLA S_i. Notice that this upper limit ($MAX_P\%$ of the price) can be agreed with the client during the SLA negotiation process.

$$Gom(\sigma(S_i)) = \begin{cases} 0 & \text{if } \sigma(S_i) \le 0 \\ MAX_P \cdot e^{-e^{-7 \cdot \sigma(S_i) + e^1}} & \text{otherwise} \end{cases} \tag{7}$$

As shown in Equation 8, $\sigma(S_i)$ function depends on the number of violations occurred for that SLA (V_i) and is parameterized with two thresholds, λ_{min} and λ_{max}, which indicate the minimum number of SLA violations in order to start paying penalties and the maximum number of SLA violations that could occur during the execution of the application, respectively.

$$\sigma(S_i) = \frac{V_i}{\lambda_{max} - \lambda_{min}} - \frac{\lambda_{min}}{\lambda_{max} - \lambda_{min}} \tag{8}$$

5 Evaluation

In this section, we compare how our metric performs with respect to the other CPU-related metrics introduced in Section 3. First, we use a small proof-of-concept workload composed of three tasks to show in detail the consequences of using each metric and then, we use a real workload to evaluate each metric.

5.1 Proof-of-Concept Workload

We have conducted three experiments. Each of them consists of the concurrent execution of three tasks, which are basically CPU-consuming tasks with variable CPU requirements over time, in a single node of the virtualized provider during 400 seconds. This makes the system able to dynamically change the resources allocated to each virtual machine and allows demonstrating the benefits of our metric in such a scenario. This node is a 64-bit architecture with 4 Intel Xeon CPUs at 2.6GHz and 16GB of RAM, which runs Xen 3.3.1 [22] in the Domain-0. This processor has been measured to have 5322.20 BogoMips, which corresponds to 10.4 ECUs approximately. Each task is started at a different time to simulate what could happen in a real scenario, where different customers can reach the provider at different times.

Each task has its own SLA, which describes the agreed QoS with the provider. For each experiment, a different SLA metric is used to specify this QoS. In particular, in the first experiment all tasks use the metric referred in Equation 1 (from now on denoted as *SLA Metric A*), in the second one they use the metric referred in Equation 2 (denoted as *SLA Metric B*), and in the last one they use the metric referred in Equation 6 (denoted as *SLA Metric C*).

For each experiment, we have monitored the allocated CPU to each task, its real CPU usage, and the value of the SLA metric at each moment in time. Comparing this value with the guarantee agreed in the SLA, we can know if the SLA is being violated or not. Figure 2 displays all this information. The top graphic in this figure shows the CPU assignment and usage for each task over time. These values will be the same independently of the metric used in the SLA. We have forced the provider to execute the three tasks, but there are not enough resources for fulfilling all the SLAs. This allows us evaluating how the different metrics deal with real SLA violations. The provider distributes the resources in such a way that the SLAs of *Task2* and *Task3* are never violated in these experiments, while *Task1* behavior shows all the possible situations regarding SLA management. For this reason, we focus the explanation only on *Task1*, which has negotiated an SLA with a CPU requirement of 6 ECUs (1/6 for Metric C).

As shown in the top graphic of Figure 2, at the beginning of the experiment *Task1* is assigned with the whole machine, since it is the only task running in the provider. When *Task2* arrives at the provider at second 150, the CPU is redistributed among *Task1* and *Task2* according to their requirements. Finally, at second 250, *Task3* arrives at the provider, which redistributes again the CPU among the three tasks. Regarding the CPU usage of *Task1*, it starts its execution

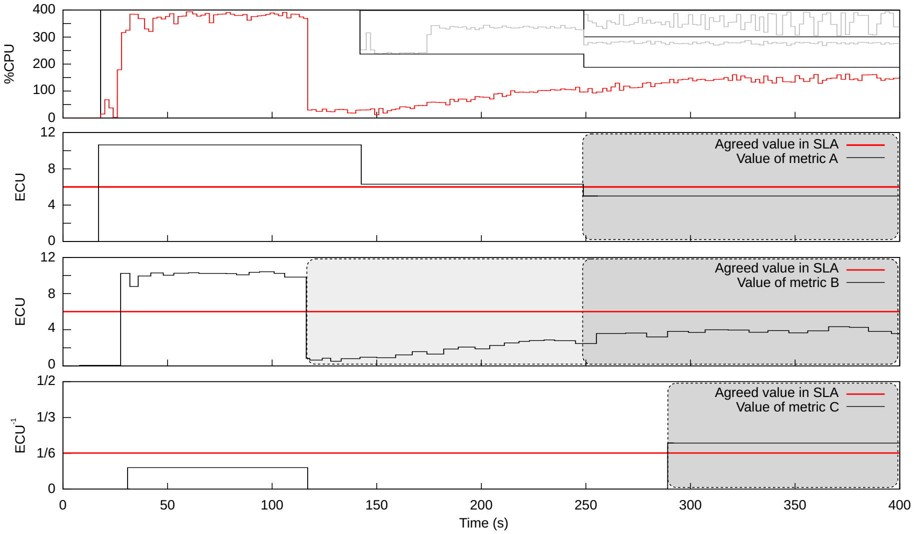

Fig. 2. CPU-Based SLA Metrics Comparison

consuming almost 400% of CPU. After 100 seconds of execution, it reduces its CPU consumption and starts using just 20% of CPU for the next 40 seconds. Then, it begins increasing its CPU usage until it consumes around 150% of CPU.

Next three graphics in Figure 2 show the value of the SLA metric over time and its agreed value in the SLA, for Metric A, B and C, respectively. As described in Section 3, *SLA Metric A* only takes into account the amount of resources assigned to the task. While this value is above of the agreed value in the SLA, SLA is fulfilled. Otherwise, an SLA violation arises. As shown in the second graphic, due to the redistribution of CPU occurred when *Task3* arrives at the provider at second 250, the value of the *SLA Metric A* for *Task1* falls below the agreed value in the SLA, thus violating it. The shaded zone identifies in the figure the interval where *Task1* is violating the SLA (from second 250 to 400). In order to avoid this, the provider would be forced to refuse executing *Task3*, even if this would be initially possible since the other tasks are not using all their allocated CPU. Notice that using this metric makes the provider unable to exploit adequately its resources, in the sense that it should always assign enough resources to fulfill the SLA, despite that tasks use them or not.

On the other side, *SLA Metric B* considers the resource usage of the tasks. Using this metric allows the provider moving freely the CPU assignment if tasks do not use all their allocated resources. Nevertheless, as shown in the third graphic, this metric can give fake SLA violations. In particular, when the CPU usage of *Task1* goes below the agreed value in the SLA (from second 115 to 400), the SLA is being violated. However, not all these violations are a responsibility of the provider. In particular, between second 115 and 250 (light gray zone in the graphic), the SLA is being violated because although the provider is giving enough resources to guarantee the SLA, the task does not use all its assigned CPU. As we have described before, this is a fake violation. On the other side, from second 250 to 400, the SLA is being violated (dark gray zone in the graphic) because the provider has not allocated enough resources to the task to fulfill its SLA, and consequently, even if it used all of them, its CPU usage would be under the SLA threshold. This is then a real violation. Notice that the value of *SLA Metric B* has some start delay with respect to the CPU usage. This is because of the different sampling rates between the SLA monitoring system and the CPU usage monitoring system.

Finally, *SLA Metric C* (the one proposed in this paper) gets the best from the two other ones, since it considers both the CPU assigned to a task and its real CPU usage. As discussed in Section 3, this metric specifies the inverse value in the SLA with respect to the other metrics. For this reason, when using this metric, an SLA violation will happen when the measured value of the metric is greater than the agreed one (1/6 in this particular experiment). As shown in the fourth graphic, this only occurs in the shaded zone, that is, from second 290 to 400. As commented before, this is the only zone where the task is using all its allocated resources and the provider has not allocated enough resources to the task to fulfill its SLA. On the other side, notice that when using this metric fake SLA violations are avoided, even when the task is not using all its allocated resources (as occurs from second 115 to 290). In these experiments, we have considered that a task is using all its allocated resources when it uses more than 70% of them (this corresponds to a 0.3 value for the α factor in Equation 6).

Summarizing, three well-differentiated phases can be appreciated in the figure. In the first one, the assigned CPU is enough to fulfill the SLA and *Task1* is using all its allocated resources. Consequently, none of the metrics is violating the SLA. In the second one, CPU allocated to *Task1* has been reduced, but the task is not using all its allocated resources. In this case, using the *SLA Metric A* causes low resource utilization because unused resources cannot be assigned to any other task without violating the SLA, while using the *SLA Metric B* causes fake SLA violations. Finally, in the last phase, the resources assignment is not enough to fulfill the SLA, and *Task1* uses all its assigned resources. In this case, all the metrics violate the SLA.

5.2 Real Workload

In this section, we evaluate the different metrics by executing a real workload. The workload is a whole day load obtained from Grid5000 [1] corresponding to Monday first of October of 2007. 215 jobs of 10 different applications are submitted during this period. The size of our real testbed is not enough to execute this workload. For this reason, we have built a simulated provider composed by 24 nodes, which mimics the behavior of the virtualized provider used in previous experiments. The number of nodes has been chosen in order to able to execute the maximum number of concurrent jobs of the workload, while still violating some SLAs to demonstrate the behavior of the different metrics.

We set up three types of nodes in the simulated provider in order to demonstrate how our proposal can deal with heterogeneous machines. The first type is the one already used in previous experiments. The other two nodes types comprise a 64-bit architecture with 4 Intel Xeon CPUs at 3GHz and 16GB of RAM, and a 64-bit architecture with 8 Intel Xeon CPUs at 2.8GHz and 16GB of RAM, which respectively correspond to 12 and 22.4 ECUs. For each task, we specify a static requirement of resources, and the dynamic resource usage it performs during its execution. Each task is executed within a VM with 2 VCPUs, since this is enough to fulfill the CPU requirements of all the applications. In order to demonstrate the impact on each metric of having dynamic CPU usage pattern over time, we have forced two of the 10 applications to use more CPU than the requested during some parts of their execution. Resource requirements and usages constitute the input of the simulator, which implements a backfilling policy taking into account these requirements in order to decide the allocated resources to each task and simulates the specified resource consumption for each task in a given time slot.

Each task has a different SLA. All of them use the same metric for specifying the CPU guarantee, but with a different agreed value, which depends on the task CPU requirement. In addition, the SLA specifies a revenue of \$0.38 per hour (we use the pricing of an Amazon EC2 medium instance [4], which has similar features to the VMs used in the experiment). The SLA is evaluated every 100 seconds, and the penalty is calculated following Equation 7 using a maximum penalty (MAX_P) of 125%. As the SLA is evaluated every 100 seconds, the maximum number of SLA violations (λ_{max}) occurs when the SLA is violated every sample. Hence, λ_{max} is equal to the number of samples, which is calculated by dividing the application execution time (T_{exec}) by the sampling period (T_{sample}). Using this information, and considering that λ_{min} is zero in our case, we can operate on Equation 8 and calculate $\sigma(S_i)$ as follows: $\sigma(S_i) = \frac{V_i \cdot T_{sample}}{T_{exec}}$.

Table 1 shows the obtained results using each one of the metrics, including the number of violations (real, fake, and total) and the final profit. For *SLA Metric C*, we have also evaluated the impact of the α factor. These results demonstrate that using our metric the provider can get the highest profit, since it avoids all the fake SLA violations and reduces the real ones when the application is not using all the allocated resources. In particular, we have reduced the incurred penalties in more than a 58% regarding *SLA Metric A* and more than a 72% compared to *SLA Metric B* when $\alpha = 0.1$. In addition, note that the lower the alpha factor is, the lower the number of SLA violations is, though the difference is not very noticeable. Nevertheless, this difference highly depends on the CPU usage of the applications. This occurs because for higher alpha values the task is considered to be using all its allocated resources during more time, which increases the probability of SLA violations imputable to the provider.

These results allow us to conclude that our metric has globally less SLA violations than the other ones, it is able to avoid fake SLA violations, and it allows the provider to freely redistribute the resources when the tasks are not using them.

Table 1. Number of SLA Violations and Profit (Real Workload)

SLA Metric	(α)	Real	Fake	Total	Profit ($)
A		3891	0	3891	231.0
B		3891	2857	6748	176.6
C	0.05	1674	0	1674	277.5
C	0.1	1782	0	1782	271.3
C	0.2	1877	0	1877	266.9
C	0.4	1884	0	1884	266.6

6 Related Work

As Cloud technology is still in its early stage, current Cloud providers only support the specification of QoS guarantees by means of SLAs in a very simple way. For instance, the SLAs supported by Amazon EC2 [4] only consider an availability metric, so-called *annual uptime percentage* [12]. It is noticeable that the violations must be monitored and claimed by the user. Furthermore, whereas Amazon EC2 offers different instances according to their ECUs, there is not any guarantee in the SLA that this computing capacity will be supplied during the whole execution of the instance.

Similarly, GoGrid also considers availability metric [14], including the availability of the storage and the primary DNS. It also offers some metrics related with network performance such as jitter, latency, and packet loss rate. Analogously, 3Tera also provides availability guarantees on its Virtual Private Datacenter [2]. Nevertheless, none of the Cloud approaches for SLA considers CPU-related metrics since they implement static provisioning of resources. As this can lead to low resource utilization, moving to dynamic provisioning is recommended to increase profit.

Until now, SLAs have been primarily used to provide QoS guarantees on e-business computing utilities. Due to this, most of the works focusing in this area [3,7,8,17,23] have used metrics of interest in these environments such as throughput, response time, and availability. None of them has included resource-level metrics as part of its solution, as we suggest in this paper.

On the other side, some effort has been carried out in the Grid environment in order to provide QoS guarantees using SLAs. Some proposals focus on service-level metrics. For instance, [10,20] propose SLA management systems for Grid environments which use the expected completion time of the jobs for defining SLA metrics. Similarly, [18] defines latency, throughput, and availability metrics and tries to minimize execution times and costs in a Grid provider. Whereas these works neglect resource-level metrics in their solution, other works define metrics for providing CPU guarantees in the SLA, but usually in a very limiting way for the resource provider. For instance, [21] allows defining the CPU count, the CPU architecture, and the CPU performance when executing a physics application on the Grid. Similarly, [19] uses metrics such as the number of processors, the CPU share, and the amount of memory. As we have discussed, when used incorrectly, some these metrics force the provider to statically overprovision CPU to the applications in order to avoid SLA violations, while others induce fake SLA violations when the applications do not use all their allocated resources. As demonstrated in the evaluation, our solution overcomes these limitations.

7 Conclusion

In this paper, we have proposed a resource-level SLA metric for specifying fine-grain QoS guarantees regarding the computing capacity of a provider. This metric overcomes the limitations of traditional CPU-related metrics. In particular, it takes into account if services are really using their allocated resources. This allows the provider to implement dynamic resource provisioning, allocating to the different services only the amount of CPU they need. In this way, better resource utilization in the provider can be achieved. In addition, dynamic resource provisioning can be accomplished while avoiding the fake SLA violations that could happen with other metrics when the customers' services do not use all the resources they have allocated. Finally, it can be used in heterogeneous machines, since it is based on Amazon's ECUs. As demonstrated in the evaluation, using this metric reduces the number of SLA violations, since it only detects those that are a real responsibility of the provider.

Although we have derived a metric for managing CPU, the idea of considering the real resource usage of the services can be used with other resources such as the memory or the network. This is part of our future work. In addition, we are working on mechanisms for supporting the inference of the resource requirements of a service to fulfill a given service-level metric.

Acknowledgments

This work is supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2007-60625 and grant AP2008-0264, and by the Generalitat de Catalunya (contract 2009-SGR-980).

References

1. The Grid Workloads Archive, `http://gwa.ewi.tudelft.nl`
2. 3Tera SLA,
   `http://www.3tera.com/News/Press-Releases/Recent/`
   `3Tera-Introduces-the-First-Five-Nines-Cloud-Computing.php`
3. Abrahao, B., Almeida, V., Almeida, J., Zhang, A., Beyer, D., Safai, F.: Self-Adaptive SLA-Driven Capacity Management for Internet Services. In: 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006), Vancouver, Canada, April 3-7, pp. 557–568 (2006)
4. Amazon EC2, `http://aws.amazon.com/ec2/`
5. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Services Agreement Specification (WS-Agreement). In: Global Grid Forum GRAAP-WG, Draft (August 2004)

6. Aneka Software, http://www.manjrasoft.com/products.html
7. Appleby, K., Fakhouri, S., Fong, L., Goldszmidt, G., Krishnakumar, S., Pazel, D., Pershing, J., Rochwerger, B.: Oceano - SLA-based Management of a Computing Utility. In: Symposium on Integrated Network Management (IM 2001), Seattle, WA, USA, May 14-18, pp. 855–868 (2001)
8. Bennani, M., Menasce, D.: Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In: 2nd International Conference on Autonomic Computing (ICAC 2005), Seattle, USA, June 13-16, pp. 229–240 (2005)
9. EU BREIN project, http://www.eu-brein.com
10. Ching, A., Sacks, L., McKee, P.: SLA Management and Resource Modelling for Grid Computing. In: London Communications Symposium (LCS 2003), London, UK, September 8-9 (2003)
11. van Dorst, W.: BogoMips mini-Howto. Tech. rep., Clifton Scientific Text Services V38 (March 2006)
12. Amazon EC2 Service Level Agreement, http://aws.amazon.com/ec2-sla/
13. Fitó, J.O., Goiri, I., Guitart, J.: SLA-driven Elastic Cloud Hosting Provider. In: 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2010), Pisa, Italy, February 17-19, pp. 111–118 (2010)
14. GoGrid Service Level Agreement, http://www.gogrid.com/legal/sla.php
15. Goiri, I., Julia, F., Ejarque, J., Palol, M., Badia, R., Guitart, J., Torres, J.: Introducing Virtual Execution Environments for Application Lifecycle Management and SLA-Driven Resource Distribution within Service Providers. In: 8th IEEE International Symposium on Network Computing and Applications (NCA 2009), July 9-11, pp. 211–218 (2009)
16. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management 11(1), 57–81 (2003)
17. Lodi, G., Panzieri, F., Rossi, D., Turrini, E.: SLA-Driven Clustering of QoS-Aware Application Servers. IEEE Transactions on Software Engineering 33(3), 186–197 (2007)
18. Menasce, D., Casalicchio, E.: Quality of Service Aspects and Metrics in Grid Computing. In: Computer Measurement Group Conference (CMG 2004), Las Vegas, USA, December 5-10, pp. 521–532 (2004)
19. Raj, H., Nathuji, R., Singh, A., England, P.: Resource Management for Isolation Enhanced Cloud Services. In: 2009 ACM Cloud Computing Security Workshop (CCSW 2009), Chicago, IL, USA, November 13, pp. 77–84. ACM, New York (2009)
20. Sahai, A., Graupner, S., Machiraju, V., van Moorsel, A.: Specifying and Monitoring Guarantees in Commercial Grids through SLA. In: 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), Tokyo, Japan, May 12-15, pp. 292–299 (2003)
21. Skital, L., Janusz, M., Slota, R., Kitowski, J.: Service Level Agreement Metrics for Real-Time Application on the Grid. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 798–806. Springer, Heidelberg (2008)
22. Xen Hypervisor, http://www.xen.org
23. Xu, J., Zhao, M., Fortes, J., Carpenter, R., Yousif, M.: On the Use of Fuzzy Modeling in Virtualized Data Center Management. In: 4th International Conference on Autonomic Computing (ICAC 2007), Jacksonville, USA, June 11-15, p. 25 (2007)