Virtualization Support for SLA-Driven Resource Distribution within Service Providers

Íñigo Goiri, Ferran Julià, Jorge Ejarque, Marc de Palol, Rosa M. Badia, Jordi Guitart, Jordi Torres
Barcelona Supercomputing Center and Universitat Politecnica de Catalunya
Jordi Girona 31, 08034 Barcelona, Spain
{inigo.goiri, ferran.julia, jorge.ejarque, marc.depalol, rosa.m.badia, jordi.guitart, jordi.torres}@bsc.es

Abstract

Resource management is a key challenge that service providers must adequately face in order to ensure their profitability. This paper describes a proof-of-concept framework for facilitating resource management in service providers, which allows reducing costs and at the same time fulfilling the quality of service agreed with the customers. This is accomplished by means of virtualization, by providing application specific virtual environments and consolidating them in order to achieve a better utilization of the providers resources. In addition, our approach implements self-adaptive capabilities for dynamically distributing the providers resources among these virtual environments based on Service Level Agreements. The proposed solution has been implemented as a part of the Semantically-Enhanced Resource Allocator prototype developed within the BREIN European project. The evaluation demonstrates that our prototype is able to react under changing conditions and avoid SLA violations by rescheduling efficiently the resources.

1 Introduction

The great popularity achieved by Internet services in business processes has encouraged the appearance of service providers that rent their resources to the enterprises on demand, providing them with a financially attractive way to host their services. Services owners pay for the actual use of provider's resources, and in return, they are provided with guarantees on resource availability and Quality of Service (QoS), which use to be expressed in the form of a Service Level Agreement (SLA).

In order to be profitable, service providers tend to share their resources among multiple concurrent applications of different customers, but at the same time they must guarantee that each application has always enough resources to meet the agreed performance goals. This task is not straight-forward, since applications typically exhibit variable resource requirements along time. For this reason, it would be desirable for the provider to implement a self-adaptive resource management mechanism, which can dynamically manage the provider's resources in the most cost-effective way (e.g. maximizing their utilization), adapting to the variable behavior of the applications while satisfying the QoS agreed with the customers.

Recently, the use of virtualization has been explored for cost reduction and easier resource management in service providers. Virtualization allows the consolidation of services, multiplexing them onto physical resources while supporting isolation from other services sharing the same physical resource, reducing in this way provider's costs and maintaining the integrity of the underlying resources and the other services. Virtualization has other valuable features for service providers. It offers the image of a dedicated and customized machine to each user, decoupling them from the system software of the underlying resource. In addition, virtualization allows agile and fine-grain dynamic resource provisioning by providing a mechanism for carefully controlling how and when the resources are used, and primitives for migrating a running machine from resource to resource if needed.

In this paper, we exploit the features of virtualization in a new approach for facilitating service providers management, which allows reducing costs and at the same time fulfilling the quality of service agreed with the customers. Our solution provides an application specific virtual environment for each application, granting full control to the application of its execution environment without any risks to the underlying system or the other applications. These virtual environments are created on demand, according to application requirements such as disk space, amount of memory, number of CPUs, required software..., and then, they are consolidated in the provider's physical resources in order to achieve better utilization (thus reducing costs).

In addition, our approach supports fine-grain dynamic resource distribution among these virtual environments based on Service Level Agreements in order to adapt to changing resource requirements of the applications. Our system guarantees to each application enough resources to meet the agreed performance goals, and furthermore, it can provide the virtual environments with supplementary resources, since free resources are also distributed among applications depending on their priority and resource demand. The system continuously monitors if the SLAs of the applications running in the provider are being fulfilled. If any SLA violation is detected, an adaptation process for requesting more resources to the provider is started.

With our solution, service providers can take advantage of all their resources by consolidating services. In addition, they can benefit from easier resource management, usage monitoring and SLA enforcement, since these tasks are implemented by means of adaptive behaviors. This enables autonomic service providers that can adapt to changes in the environment conditions without any additional effort to the system administrator.

The components described in this paper are part of the Semantically-Enhanced Resource Allocator (SERA) prototype [5] developed within the BREIN European IST Project [4]. BREIN aims to develop a framework to enhance business relationships among service providers and their customers using challenging technologies such as agents and semantics. The SERA prototype enables resource allocation depending on the information given by service providers with regard to the level of preference (according to business goals) of their customers and on the requirements of their tasks. The allocation process is enhanced by using agents, semantics and virtualization.

The paper is organized as follows: Section 2 presents an overview of the SERA. Section 3 introduces our proposal for managing virtual machines and dynamically provisioning resources. Section 4 presents our SLA management strategy, including SLA monitoring and SLA enforcement. Section 5 describes the experimental environment and the evaluation. Section 6 presents the related work. Finally, Section 7 presents the conclusions of the paper and the future work.

2 SERA Overall Architecture

This section gives an overview of the architecture of the SERA, describing the main components and their interactions. Each component contains an agent and a core. The agent wraps the core functionalities by means of a set of behaviors which basically call methods from this core. The agents are in charge of the communication between components. In addition, their implicit reactiveness is used to implement the self-adaptive behavior of the system, that is, being aware of the system performance and status variations, and coordinating the reaction to these variations (e.g. reaction to an SLA violation).

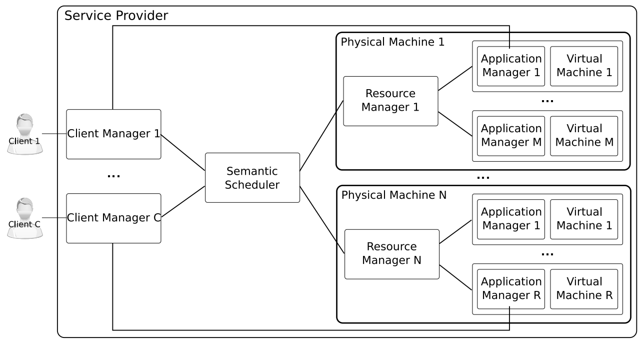

Figure 1. Architecture of SERA prototype

Figure 1 shows the main components of the SERA, whose functionality is herewith described. The Client Manager (CM) manages the client's task execution by requesting the required resources and by running jobs. In addition, it makes decisions about what must be done when unexpected events such as SLA violations happen.

The Semantic Scheduler (SeS) allocates resources to each task according to its requirements, its priority and the system status, in such a way that the clients with more priority are favored. Allocation decisions are derived with a rule engine using semantic descriptions of tasks and physical resources. These resource descriptions are automatically generated from the system properties and stored in a Semantic Metadata Repository (SMR) when the machine boots. This repository comes with a set of services for registering, storing and publishing the semantic descriptions.

The Resource Manager (RM) creates virtual machines (VM) to execute clients' tasks according to the minimum resource allocation (CPU, memory, disk space...) given by the SeS and the task requirements (e.g. needed software). Once the VM is created, the RM dynamically redistributes the remaining resources among the different tasks depending on the resource usage of each task, its priority and its SLA status (i.e. is it being violated?). This resource redistribution mechanism allows increasing the allocated resources to a task by reducing the assignment to other tasks that are not using them.

Finally, the Application Manager (AM) monitors the resource usage and the SLA parameters in order to evaluate if an SLA is being violated. An SLA violation can be solved by requesting more resources to the RM. If the RM cannot provide more resources, the AM will forward the request to the CM.

Figure 2 shows the task lifecycle in the SERA and the interaction among the different components. Initially, at boot time, every component stores its semantic description in the

SMR. An interaction starts when a task arrives at the system and a CM is created in order to manage its execution. The CM preselects potential nodes for running the task querying the SMR and registers the task description. Then, it requests a time slot to the SeS for the task (1). In this stage, the SeS uses the metadata stored in the SMR to infer in which node the task will be executed. At this point the SeS informs the CM whether the task has been successfully scheduled or canceled. When the time to execute the task arrives, the SeS contacts with the RM in charge of the node where the task has been allocated and requests the creation of a VM for executing the task (2).

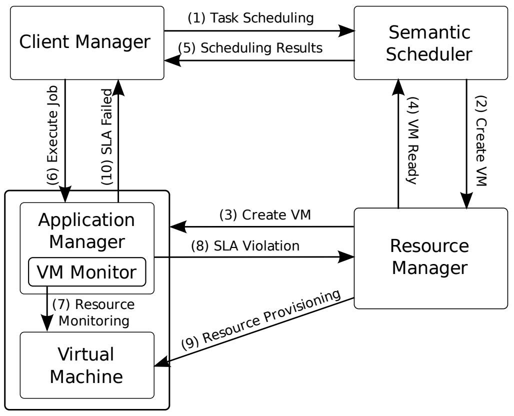

Figure 2. Task lifecycle

When the RM receives the SeS request, it creates a VM and an AM that will monitor the SLA fulfillment for this task (3). Once the VM is created, the SeS is informed (4) and it forwards the message to the CM indicating the access information to that VM (5). At this point, the CM can submit the task to the newly created VM (6). From this moment, the task is executed in a VM which is being monitored by the AM in order to detect SLA violations (7). If this occurs, the AM requests more resources to the RM (8), trying to solve the SLA violation locally to the node (9). This request for more resources will be performed as many times as needed until the SLA is not violated any more or the RM informs the AM that the requested resource increment is not possible. In the latter case, since the SLA violation cannot be solved locally, the AM informs the CM about this situation (10). In this case, the CM should decide to resubmit the task with higher resource requirements or notify the client if the resubmission fails.

This paper focuses mainly in the functionality and implementation of the RM and the AM components, which are described in the following sections. Additional description of the other components can be found in [5].

3 Resource Manager

The Resource Manager (RM) is composed by its corresponding agent and core. There is one RM instance per physical machine in the service provider. Once the RM is created and fully started, it waits for requests from the SeS. When a new request arrives, the RM checks if it is possible to create a new VM with the specified features and it informs the SeS about the success/failure of this operation. Virtualization is our system is supported by using Xen [22].

3.1 Management of VMs lifecycle

The creation of a new VM requires the following steps: downloading and creating the guest operating system (a Debian Lenny through debootstrap for this prototype), copying extra software needed by the client in an image that will be automatically mounted in the VM, creating home directories and swap space, setting up the whole environment, packing it in an image, and starting the VM. Once the VM has completely started, the guest operating system must be booted. After this, the additional software needed by the client must be also instantiated (if applicable). These phases can be clearly appreciated in Figure 5, which is presented in Section 6.

From this description, one can derive that this process can have two bottlenecks: the network (for downloading the whole system) and the disk (for copying applications and creating system images, approx. 1GB of data). The network bottleneck has been solved by creating a default image of the system with no settings, and copying it for each new VM. This almost eliminates the downloading time (base system is only downloaded once and can be reused for each new VM), but contributes to the disk bottleneck. The disk bottleneck has been solved by adding a second caching system that periodically copies the default image and the images with the most commonly used software to a cache space. Finally, the RM has only to move these images (just an i-node change) to the final location when a new VM is created. Using these caching techniques, the complete creation of a VM has been reduced from up to 40 seconds to an average time of 7 seconds. More details about the VM creation times will be shown in Section 6.

Additionally, when a task finishes or the SeS decides that this task should be rescheduled or canceled, the VM must be destroyed. This includes killing the associated AM, and the redistribution of the resources freed by this VM among the other VMs (see Section 3.2).

3.2 Resource distribution among VMs

The RM is also responsible of distributing the providers physical resources among the VMs. The goal is to max-

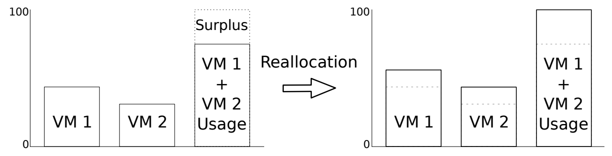

Figure 3. Surplus resource distribution

imize physical resources utilization, while fulfilling the SLAs. In order to accomplish this, the SeS provides the RM with two parameters for each VM, namely the minimum resource requirements of the VM and the initial priority of this VM, which corresponds to the priority for the service provider of the customer executing in this VM (e.g. Gold, Silver, etc.).

For each VM, the RM guarantees that its minimum resource requirements are met during the whole VM lifetime. Surplus resources that are not allocated to any VM are dynamically redistributed among VMs according to their resource usage and the fulfillment status of the SLAs (as shown in Figure 3). In this way resource wasting is avoided and the applications are provided with better service.

The surplus resources are redistributed among the VMs according to their dynamic priority. This priority initially corresponds to the priority set by the SeS and can be dynamically increased by the AM to apply for more resources if the SLA is being violated. Any dynamic priority change induces the RM to recalculate the resource assignment of the VMs according to the following formula (where p_i is the priority of client i) and bind the resources to the VMs.

$$R_{assigned}(i) = R_{requested}(i) + \frac{p_i}{\sum_{j=0}^{N} p_j} \cdot R_{surplus}$$

Current implementation is able to manage CPU and memory resources. CPU management is straight-forward by using the Xen Scheduler credit policy. This policy allows specifying the maximum amount of CPU assigned to a VM by defining scheduling priorities. For example, in a platform with 4 CPUs (i.e. 400% of CPU capacity) and two VMs, one with a priority of 6 and the other with a priority of 4, the first could take at most the 240% of CPU, while the other could take at most the rest 160% of CPU. The scheduling priorities can be set using the XenStat API.

On the other side, there are some limitations for dynamic memory management using VMs. In Linux systems, the mapping of physical memory is done at boot time. Once the guest system has booted, if the amount of memory allocated to the VM is reduced, the guest system adapts to this reduction automatically. Nevertheless, when assigning to the VM more memory than the initially detected by the guest system, Linux does not make it available to the user. It would be necessary to restart the guest system to make all this memory available. In order to overcome this limitation, the RM creates all the VMs with the maximum amount of memory possible and then it reduces the amount of allocated memory to the value indicated by the SeS.

4 Application Manager

The Application Manager (AM) has two main responsibilities. On one side, it enables the execution of the task into the VM created by the RM explicitly for this task. This is done by means of a Globus Toolkit 4 (GT4) [7] deployed in each VM. The GT4 is configured during the VM creation and started at the VM boot time, in such a way that the CM can easily submit the task to the VM and check its state using the GT4 interface. On the other side, the AM is in charge of monitoring the resources *provided to* and *used by* the VM ensuring the fulfillment of its SLA. There is one AM instance per application running in the service provider, thus several AM instances can exist per physical machine.

4.1 SLA description

Each application has its own SLA, described in XML using both WS-Agreement [9] and WSLA [16] specifications, as done in TrustCoM [21]. The SLA includes two simple metrics: the amount of memory used and a performance metric which intends to compute the real usage of the CPU (see the definition below). An application will have the amount of cycles/sec specified in the SLA whereas it uses all the assigned CPU. If the application is not using all the resources then the SLA will be always fulfilled. The SLA will be violated only when the application is using all the assigned resources and these resources are not enough. The performance metric for the CPU usage is defined as $\frac{\text{Used CPU}}{100} \cdot \text{CPU frequency}$. This is a very simple metric used to test the viability of our proposal, assuming that the pool of machines is homogeneous (as occurs in our testbed). Next version of the prototype will support heterogeneous machines.

When defining SLA metrics, it is desirable that they can be defined through average values, since otherwise the evaluation of the SLA metrics can depend more on when they are measured than in the metrics themselves. This can be accomplished through the SLA specifications, which allow defining the window size of the average and the interval between two consecutive measures. The following code gives an example showing how we have used these capabilities in our SLAs. Notice that we have defined a window size of 10, and an interval of 2 seconds.

```
...
<wsla:Schedule name="twoSecSchedule">
  <wsla:Period>
    <wsla:Start>
```

```
        2005-12-31T00:00:00
      </wsla:Start>
      <wsla:End>
        2008-12-31T00:00:00
      </wsla:End>
    </wsla:Period>

    <wsla:Interval>
      <wsla:Seconds>2</wsla:Seconds>
    </wsla:Interval>
</wsla:Schedule>
...
<wsla:Metric name="average_process_Cpu_Load"
      type="double" unit="percent">
  <wsla:Source>
    Application_monitor
  </wsla:Source>
  <wsla:Function resultType="double"
    xsi:type="wsla:Mean">
    <wsla:Metric>
      process_cpu_time_serie
    </wsla:Metric>
  </wsla:Function>
</wsla:Metric>

<wsla:Metric name="process_cpu_time_serie"
      type="double" unit="percent">
  <wsla:Source>
    Application_monitor
  </wsla:Source>
  <wsla:Function resultType="TS"
    xsi:type="TSConstructor">
    <wsla:Schedule>
      twoSecSchedule
    </wsla:Schedule>
    <wsla:Metric>
      process_Cpu_Load
    </wsla:Metric>
    <wsla:Window>10</wsla:Window>
  </wsla:Function>
</wsla:Metric>
...
```

4.2 SLA monitoring

The AM includes a subsystem for monitoring the metrics defined in the SLA, which is implemented using daemons running in the Xen Domain-0. There is one daemon per VM, which obtains all the information referent to this VM via XML-RPC calls to the Xen API.

These daemons cannot run inside the VMs for two reasons. Firstly, doing this would consume part of the assigned resources to the task execution, charging to the customer the cost (in CPU and memory) of this monitoring. Secondly, the monitoring system cannot take real measures inside the VM: this can be only accomplished by measuring from the Xen Domain-0. We have tried other monitoring options (e.g. Ganglia [6]), but since they do not have explicit support for virtualization, the obtained measures were not correct.

4.3 SLA enforcement

Described proposal assumes that the SLA negotiation between the customer and the provider has been carried out previously, being our prototype responsible of guarantying the fulfillment of the agreed SLA by means of adequate resource allocation. The agreed SLA is attached to the task execution request that arrives at the RM. This SLA is assigned to the AM that monitors the VM which is going to execute the task.

The agent within this AM executes the SLA enforcement cycle shown in Figure 4. Notice that, the needed reactiveness to SLA violations can be easily accomplished using agent behaviors. The cycle is executed every second. This forces the minimum granularity of the interval between two consecutive measures for each metric in the SLA (see Section 4.1) to be also 1 second.

The *Direct Measurer* component gets the values from the monitoring subsystem, controlling the measurement intervals for each metric in the SLA and ensuring the refresh of the measures on correct time. When the new values arrive at the *Measurer Sched* component, it checks if any metric has updated its value. In this case, the *Measurer Sched* recalculates the top level metric defined in the SLA and it compares the result with the agreed value specified in the SLA. If the SLA is fulfilled, the *Measurer Sched* waits until the next iteration, otherwise the SLA violation protocol starts.

The first step in the SLA violation protocol is requesting more resources to the RM (by increasing the VM dynamic priority). If the node has surplus resources that can be assigned to that VM, the RM will redistribute them as described in Section 3.2 and the SLA cycle will start again. If all the physical resources are already allocated, the AM will contact the CM to communicate the SLA violation. When the CM receives this notification, it must decide what to do with the task that is violating its SLA, taking into account the customer's priority and the task deadline. This is currently not implemented, but possible actions include continuing the execution besides the SLA violation (if this is acceptable for the customer), modifying the resource requirements of the task and reschedule it (if task deadline permits the rescheduling), or canceling the task.

Notice that, since the RM always guarantees the minimum amount of resources specified by the SeS when it requests the creation of a VM, the SLA can only be violated when this minimum amount of resources are not enough to fulfill the SLA. This can occur when running an application with variable resource requirements along time (e.g. a web server), which receives an unexpected workload, or as a result of an error in the SeS inference process for estimating the resources. The latest situation has been assumed in Section 6 to test the functionality of the SERA.

Using the adaptation mechanism described in this sec-

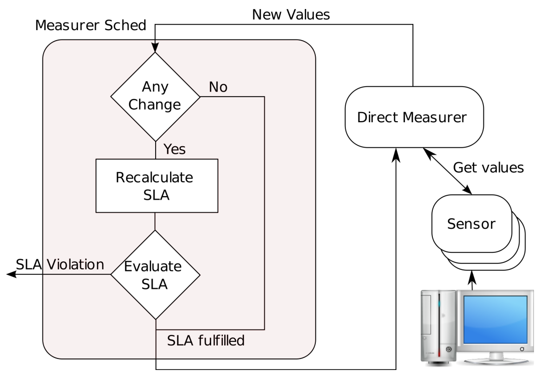

Figure 4. SLA enforcement cycle

tion, the system is able to manage itself avoiding as much as possible the SLA violations. Of course, more intelligence could be applied in this part. For example, we are planning to incorporate economic algorithms taking in account the penalizations for violating the SLA, in order to get the best configurations in terms of profit.

5 Experimental Environment

As commented in Section 2, RM and AM are part of a bigger system that enables semantic resource allocation in a virtualized environment. In order to evaluate our proposal, the whole system needs to be executed, although results presented in this paper concentrate mainly in the described components. The main technologies used in the SERA prototype include Ontokit [17], which is a Semantic OGSA implementation, for implementing the SMR; the Jena 2 framework [11] for supporting inferences and the management of the semantic metadata; and Jade [10] as the agent platform. Xen [22] is used as virtualization software.

Our experimental testbed consists on two machines. The first one is a Pentium D with two CPUs at 3.2GHz with 2GB of RAM that runs the SeS, the CMs and the SMR. The second machine is a 64-bit architecture with 4 Intel Xeon CPUs at 3.0GHz and 10GB of RAM memory. It runs Xen 3.1 and a RM executes in the Domain-0. These machines are connected through a Gigabit Ethernet.

Most part of the software is written in Java and runs under a JRE 1.5, except the scripts that manage the creation of the virtual machines, which are written in Bash script, and some libraries used for accessing Xen, which are in C.

6 Experimental Evaluation

This section presents the evaluation of our proposal, which includes the quantification of the time needed to create a VM and a proof-of-concept experiment for demonstrating the functionality of the whole SERA prototype, but focusing on the SLA-driven dynamic resource distribution. The applications used in the experiments simulate scientific applications with high CPU consumption.

6.1 VM creation performance

This section provides some indicative measurements about the time needed to create a VM and make it usable for a customer's application, and the benefit of our cache systems for reducing the VM creation time compared with the default approach.

As described in Section 3, the creation of a VM implies downloading and creating a default base system image with debootstrap. This requires around 150 seconds, but it is only done once and can be reused for each new VM. The second caching level, which consists of pre-copying the default and the software image, takes approximately 60 seconds per VM. Finally, applying whole caching of each disk image (including home and swap spaces) needs 13.5 seconds. Using both caching systems, an image can be created in only 2 seconds. The different times needed for the creation of a typical VM are summarized in Table 1. Notice that, once the image is ready, it must be loaded, which needs 4 seconds. According to this, the total time needed to have a full configured system started is less than 7 seconds, when taking advantage of the whole caching system.

Action	Time
Create and download default system image	152.6
Create cached base system image	59.1
Create cached base software image	13.9
Create image using caching systems	2.3
Load image	4.4
Total time for running an image	6.7

Table 1. VM creation times

Nevertheless, the above VM creation time does not include the time that is needed by the guest operating system to boot and be available to the user. In addition, the time needed for instantiating installed software must be also considered. All these times can be appreciated in Figure 5, which shows the CPU usage of a given VM during its whole lifetime (from the VM creation to the VM destruction). During the phase A, the Xen Domain-0 creates the VM. This spends almost one CPU. During the phase B, the guest operating system is booted (first peak in the CPU usage graph) and then the GT4 is started, which includes certificates creation and deployment of the container (second peak in the CPU usage graph). At this point, the customer's task can be submitted, executing during the phase C. Finally, during the phase D, the Xen Domain-0 destroys the

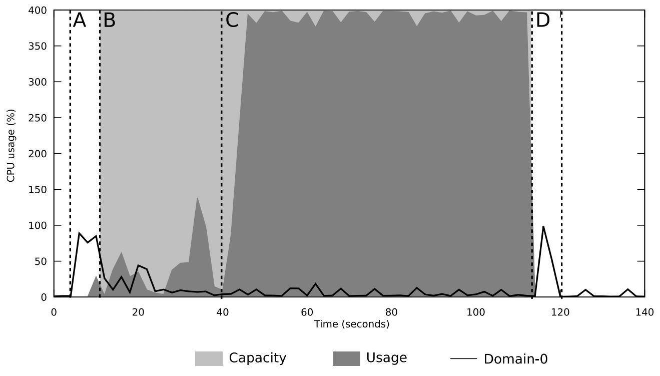

Figure 5. VM lifecycle

VM. Notice that the CPU consumption of the Xen Domain-0 is only noticeable during the creation and destruction of the VM.

The results in this figure confirm that the creation of a VM takes around 6 seconds, while the guest system boot and the GT4 start take around 30 seconds. According to this, the full creation of the VM takes around 36 seconds (from the moment that the RM receives the request until the moment when the VM is fully functional and the customer's task can be executed). In addition, VM destruction takes also 6 seconds.

6.2 SLA-driven resource distribution

This experiment demonstrates how resources are dynamically reallocated among applications and how the system is able to detect and resolve an SLA violation from one of the applications and reallocate resources (memory and CPU) until all the SLAs are fulfilled.

The experiment consists of running a total amount of three tasks with different requirements and different SLAs within three different VMs located in the same machine. Table 2 describes for each task: its requested CPU (i.e. the value provided by the SeS) (*REQ*), and the agreed CPU metric in the SLA (*SLA*).

The amount of CPU allocated to each VM is quantified using the typical Linux CPU usage metric (i.e. for a computer with 4 CPUs, the maximum amount of CPU will be 400%). For simplicity, all the measures and figures are referred only to CPU.

Figure 6 displays the execution of the three tasks executed in the test. The first three plots show the allocated CPU for this particular VM, the CPU really consumed in each moment by the VM and a horizontal line that indicates the SLA threshold. If at any moment the consumed CPU is the same then the allocated CPU and this value is lower than the SLA line, it means that the SLA is being violated.

	CPU	
	Req (%)	SLA
Task1	100	100
Task2	100	190
Task3	105	105

Table 2. Description of tasks

Notice that, SLA violations occur only if the task is using all the allocated resources, and despite this, the SLA threshold is not respected. Finally, the fourth plot corresponds to the consumed CPU by the Xen Domain-0, and it shows the CPU costs of creating and destroying VMs and of reallocating resources among them.

In the Figure 6 we can distinguish three situation of the system:

Zone A. This is the initial part of the experiment. Task1, which requires 100% of CPU (2), arrives at the system. When the request for creating the first VM is received, the RM has no other VM to manage, hence, it gives the whole machine (400% of CPU) to this newly created VM.

Task2, with a CPU requirement of 100% of CPU, is sent to the system. Task2 represents a special case: we assume that there has been an error when estimating the minimum amount of resources needed to fulfill the SLA, so that they are not enough to fulfill the SLA. In this case the SLA needs 190% of CPU to be fulfilled, but the CPU requested by the SeS for this VM is only 100%. When this second VM is created, the total amount of CPU allocated in the machine is 200%, so there will be 200% of CPU free, which is distributed among the two VMs (100% for each). At this moment, any SLA is being violated, as both Task1 and Task2 have 200% of CPU. At the end of Zone A we have two applications sharing the whole machine.

Zone B. In this zone, Task3, which requires 100% of CPU, is started, so the allocated CPU will be 300% of 400%. In this situation, the surplus resources are also equally shared among all the VMs, and, as we can see at the beginning of Zone B, all the tasks have 133% of CPU. Having only 133% of CPU, Task2 violates its SLA (notice that at the center of Zone B, the assigned CPU is under the SLA threshold). This starts the SLA violation protocol, which reallocates the CPU progressively until all the SLAs are fulfilled. We can see how the surplus CPU assigned to Task1 and Task3 is moved to Task2 until its allocated CPU is over the SLA threshold.

Zone C. This is the final zone, where all the SLAs are again

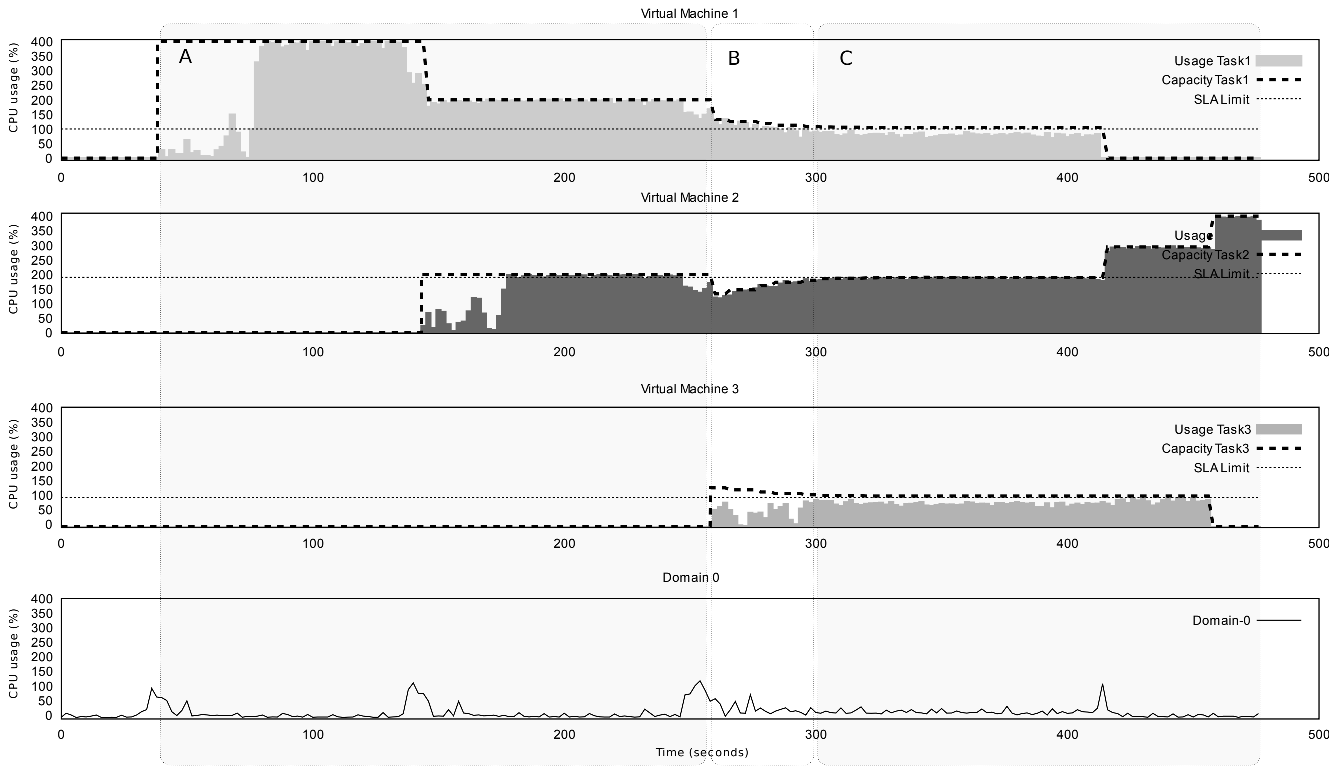

Figure 6. CPU allocation and consumption

within their thresholds. Notice that when a task finalizes its execution, its allocated resources are freed and then redistributed among the other tasks.

Obviously, if a fourth application arrived at this machine and allocated all the surplus resources, the SLA for the Task2 would be violated again, and in this case, there would not be enough free resources to solve this violation. In this situation, the AM would communicate this situation to the CM, in order to reschedule the task which is violating the SLA in another machine.

Finally, Figure 7 gives a global vision of Figure 6 showing how the physical machine is partitioned along time among the three executed tasks and how the RM (contained in the Xen Domain-0) uses the CPU in every moment. This figure shows that the whole machine is always assigned, maximizing resource utilization.

7 Related Work

The literature includes a considerable amount of works proposing solutions for resource management in service providers. Some of these solutions are driven by SLAs, as in our case. For example, in [3], the authors combine the use of analytic predictive multiclass queuing network models and combinatorial search techniques to design a controller for determining the required number of servers for each application environment in order to continuously meet the SLA goals under a dynamically varying workload. Another example is Oceano [2], which is a SLA-driven prototype for server farms, which enables the dynamic moving of servers across clusters depending on the customer changing needs. The addition and removal of servers from clusters is triggered by SLA violations. Also in [15], which presents a middleware for J2EE clusters that optimizes the resource usage to allow application servers fulfilling their SLA without incurring in resource over-provisioning costs. This resource allocation is done adding or removing nodes from the cluster.

Lately, some works have exploited virtualization capabilities for building their solutions. On one hand, virtualization has been used to facilitate system administration and provide the users with dedicated and customized virtual working environments, making more comfortable their work. For example, VMShop [14] is a virtual management system which provides application execution environments for Grid Computing. It uses VMPlant to provide automated configuration to meet application needs. VMPlant also allows the creation of flexible VMs that can be efficiently deployed (by implementing a caching-based deploy-

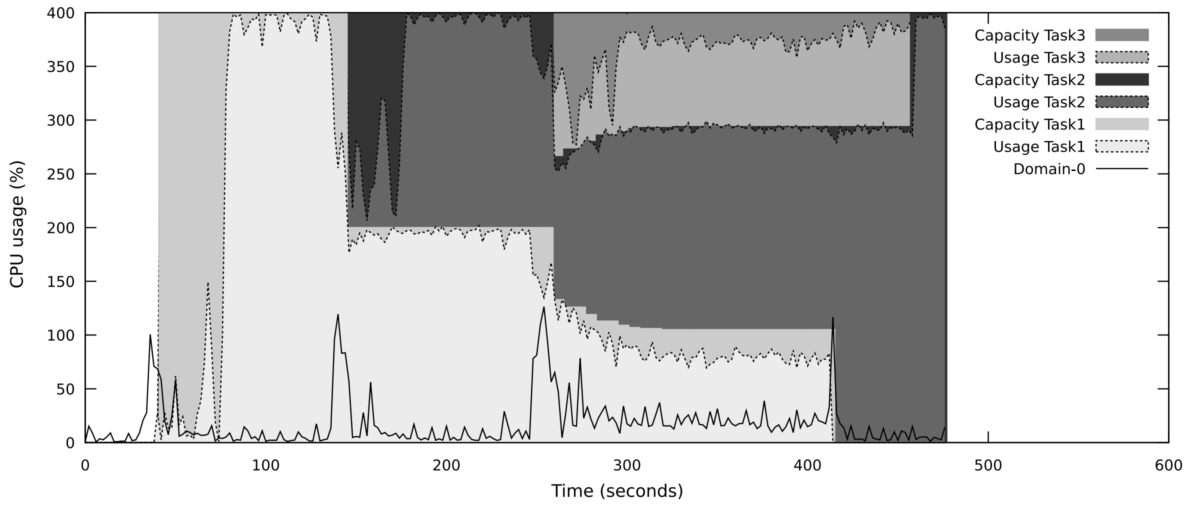

Figure 7. CPU allocation between VMs

ment) across distributed Grid resources. In addition to this, it is typical that these virtual environments can be scheduled among different nodes by using virtualization features such as pausing and migration, as occurs in Globus Virtual Workspace [13] and SoftUDC [12]. Additionally, the latter adds an efficient shared storage between nodes located in different locations. This project provides the capability of sharing resources of different organizations and solves problems such as sharing data between separated clusters.

On the other hand, while the above proposals deal only with the global scheduling of VMs between nodes, other works have also used virtualization to enable fine-grain dynamic resource distribution among VMs in a single node. For instance, [19] develops an adaptive and dynamic resource flowing manager among VMs, which uses dynamic priorities for adjusting resource assignation between VMs over a single server for optimizing global machine performance. [18] introduces an adaptive resource control system (implemented using classical control theory) that dynamically adjusts the resource shares to VMs, which contain individual components of complex, multi-tier enterprise applications in a shared hosting environment, in order to meet application-level QoS goals. [20] takes advantage of virtualization features to collocate heterogeneous workloads on any server machine, thus reducing the granularity of resource allocation. Finally, [8] has developed a new communication-aware CPU scheduling algorithm that improves the performance of a default Xen monitor by enabling the underlying scheduler being aware about the behavior of hosted applications.

Our work proposes a more general and extensive solution for managing service providers by joining in a single framework the creation of application specific virtual execution environments on demand, the global resource allocation among nodes, and the SLA-driven dynamic resource redistribution at node level (based on the redistribution of surplus resources). Some other works combine some of these functionalities, albeit none of them provides all our facilities. In particular, [1] proposes a dynamic capacity management framework for virtualized hosting services, which is based on an optimization model that links a cost model based on SLA contracts with an analytical queuing-based performance model. However, this work does not support either the creation of VMs on demand or the two-level resource allocation. In addition, the evaluation does not use a working implementation of the proposed system, but a discrete event simulation. Similarly, [23] presents a two-level autonomic resource management system for virtualized data centers that enables automatic and adaptive resource provisioning in accordance with SLAs specifying dynamic tradeoffs of service quality and cost. A novelty this approach is the use of fuzzy logic to characterize the relationship between application workload and resource demand. However, this work does not support the creation of VMs on demand either.

8 Conclusions and Future Work

This paper has described a working prototype of a framework for facilitating resource management in service providers, which is part of the Semantically-Enhanced Resource Allocator developed within the BREIN European project. Described solution exploits the well-known features of virtualization for providing application specific virtual environments, granting in this way full control to the applications of their execution environment without any risks to the underlying system or the other applications. These virtual environments are created on demand and consolidated in the provider's physical resources. This allows the provider to use better its resources and reduce costs.

In addition, our approach supports fine-grain dynamic resource distribution among these virtual environments based on Service Level Agreements (encoded using a real SLA specification). The system implements a self-adaptive behavior: each application receives enough resources to meet the agreed performance goals, and free resources can be dynamically redistributed among applications when SLA violations are detected.

We have presented experimental results demonstrating the effectiveness of our approach. These experiments show that application specific VMs can be ready to be used in a short period of time (around 7 seconds). In addition, the evaluation demonstrates that our system is able to adapt the resource allocations under changing conditions while fulfilling the agreed performance metrics and solve SLA violations by rescheduling efficiently the resources.

Although current prototype has a pretty good functionality, we are planning some improvements which include adding more complex policies to the RM based on economic parameters (e.g. rewards, penalties), incorporating the pausing/resuming mechanism for pausing tasks when others need more resources, and migrating tasks when there is an SLA violation and there are not enough resources on the local machine to solve it. In addition, we are extending the system to manage additional resources such as the network and the disk, and for considering the Xen Domain-0 CPU usage in the resource allocation decisions.

References

[1] B. Abrahao, V. Almeida, J. Almeida, A. Zhang, D. Beyer, and F. Safai. 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, 2006*, pages 557–568, 2006.

[2] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Oceano - SLA-based Management of a Computing Utility. In *IFIP/IEEE Symposium on Integrated Network Management (IM 2001), Seattle, Washington, USA, May 14–18, 2001*, pages 855–868, 2001.

[3] M. Bennani and D. Menasce. Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In *2nd International Conference on Autonomic Computing (ICAC'05), Seattle, Washington, USA, June 13–16, 2005*, pages 229–240, 2005.

[4] EU BREIN project. http://www.eu-brein.com.

[5] J. Ejarque, M. de Palol, F. Julià, I. Goiri, J. Guitart, R. M. Badia, and J. Torres. Using Semantics for Enhancing Resource Allocation in Service Providers. Technical Report UPC-DAC-RR-2008-3, 2007.

[6] Ganglia Monitoring System. http://ganglia.sourceforge.net/.

[7] Globus Toolkit. http://globus.org/toolkit/.

[8] S. Govindan, A. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Xen and Co.: Communication-aware CPU Scheduling for Consolidated Xen-based Hosting Platforms. In *3rd International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE'07), San Diego, California, USA, June 13–15, 2007*, pages 126–136, 2007.

[9] GRAAP Working Group. Web Services Agreement Specification (WS-Agreement), Version 2005/09, Global Grid Forum. Technical report, 2005.

[10] Java Agent DEvelopment Framework. http://jade.tilab.com/.

[11] Jena Semantic Web Framework. http://jena.sourceforge.net/.

[12] M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton, and F. Gittler. SoftUDC: a Software-based Data Center for Utility Computing. *Computer*, 37(11):38–46, 2004.

[13] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual Workspaces in the Grid. In *11th Europar Conference, Lisbon, Portugal, September, 2005*, pages 421–431, 2005.

[14] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In *2004 ACM/IEEE conference on Supercomputing (SC '04)*, page 7, Washington, DC, USA, 2004. IEEE Computer Society.

[15] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini. SLA-Driven Clustering of QoS-Aware Application Servers. *IEEE Transactions on Software Engineering*, 33(3):186–197, 2007.

[16] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. Web Service Level Agreement (WSLA) Language Specification, Version 1.0, IBM Corporation. Technical report, 2003.

[17] EU OntoGrid project. http://www.ontogrid.net.

[18] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive Control of Virtualized Resources in Utility Computing Environments. *ACM SIGOPS Operating Systems Review*, 41(3):289–302, 2007.

[19] Y. Song, Y. Sun, H. Wang, and X. Song. An Adaptive Resource Flowing Scheme amongst VMs in a VM-Based Utility Computing. In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, pages 1053–1058, October 2007.

[20] M. Steinder, I. Whalley, D. Carrera, I. Gaweda, and D. Chess. Server Virtualization in Autonomic Management of Heterogeneous Workloads. In *10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07), Munich, Germany, May 21–25, 2007*, pages 139–148, 2007.

[21] The TrustCoM Project, Priority IST-2002-2.3.1.9. 30 Apr. 2006. http://www.eu-trustcom.com.

[22] Xen Hypervisor. http://www.xen.org.

[23] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the Use of Fuzzy Modeling in Virtualized Data Center Management. In *4th International Conference on Autonomic Computing (ICAC 2007), Jacksonville, Florida, USA, June 11–15, 2007*, page 25, 2007.