Using Semantics for Resource Allocation in Computing Service Providers

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

Abstract

Service providers (SP) business goals require an efficient management of their computational resources in order to perform provisioning, deployment, execution and adaptation which traditionally require human intervention. The Semantically-Enhanced Resource Allocator (SERA) is a framework designed to obtain an efficient autonomous service provider management while fulfilling customers’ requests. While the efficient dynamic resource management is obtained by means of virtualization and agents, in this paper, we focus on how semantics can help on the task scheduling and resource allocation. In SERA, tasks and resources are semantically described following a resource ontology and resource assignments are inferred by means of a set of rules that define the scheduling policies taking into account the SP business goals.

1 Introduction

The BREIN European project [2] has as objective bringing the concepts developed in Grid research towards a more business-centric model, enhancing the system with methods from artificial intelligence, agent systems, semantic web, etc. In the scenario considered in BREIN, a customer submits a goal to be achieved, which is translated into an abstract workflow. For each step a business collaboration will be started to find out the service suppliers that could perform each step.

The aim of our work deals with the case of a SP which is able to provide different type of computation services, i.e. transactional applications and long running batch jobs. This implies not only an heterogeneous workload but also different type of SLAs agreed with the different customers. Additionally, the SP should try to maximize its profit, by making an optimal use of the resources (reducing the penalties incurred by not meeting the SLA terms), maximize the customer satisfaction, and in case of conflict, penalize first customers with less priority.

Under these premises, a Semantically-Enhanced Resource Allocator (SERA) has been implemented that benefits from technologies such as semantics, agents and virtualization. It provides a framework which is able to schedule the customer requests taking into account not only the SLA terms, but also the state of the SP resources and the SP level of preference for each customer. Besides, we want to provide the system with the ability of re-scheduling customers’ requests based on their priority and considering advanced reservations. With this objective, the whole environment is semantically described in order to able the resource assignments inference based on a set of rules. Additionally, virtualization is used as a mean for providing a customized and isolated virtual environment for each application supporting a fine-grain dynamic resource distribution in order to adapt to changing application requirements. The structure of the paper is the following: section 2 summarizes the related work that can be found in the literature, section 3 describes the SERA architecture and section 4 presents how the semantic technologies are used for resource allocation. Some experimental results are presented in section 5 and section 6 concludes the paper.

2 Related Work

Traditionally in grids, each organization published their resource properties and application requirements using their own language and meanings making interoperation very expensive. The Semantic Grid [13] tackles this problem introducing semantic web technologies in Grids. The EU FP6 OntoGrid project [9] proposes a reference model to support semantics in Grids called Semantic OGSA [4] which consists of a set of OGSA compliant services in charge of managing semantic descriptions of grid entities and reasoning resources. Regarding resource allocation, [5] and [8] present two different ontology-based resource matchmaking algorithms implemented by a set of rules which identify the resources which fulfill some requirements. The Phosphorus project [11] use semantics to automatically select resources. However, the resource matchmaking is only a part of the resource allocation and job scheduling. In this paper, semantics are used in the whole resource allocation
process, that is, resource matchmaking and scheduling of tasks in the selected resources taking into account not only the task requirements, but also business parameters. Finally, we also want to mention other previous work about semantic support for scheduling and resource coordination presented in [10] and [1]. We have partially used their results to build the SERA.

### 3 Overall Architecture

This section gives an overview of the architecture of the SERA. Each component contains an agent and a core. The agent wraps the core functionalities by means of a set of behaviours which basically call methods from this core. Moreover, they are aware of the system status variations and coordinate the system adaptation (e.g. SLA violation).

The Client Manager (CM) is in charge of the client’s task execution and takes 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, priority and the system status, in such a way that the customers with more priority are favored. The Resource Manager (RM) creates virtual machines (VM) to execute clients’ tasks according to the minimum resource allocation (CPU, memory, disk...) 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. 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 in order to evaluate if an SLA is being violated. Finally, the Semantic Metadata Repository (SMR) maintains the semantic description of the components.

Figure 1 shows the task lifecycle in the SERA and how the different components interact. Initially, components store its semantic description in the SMR. An interaction starts when a task arrives to the system and a CM is created to manage its execution. The CM preselects potential nodes for running the task querying the SMR and registers the task description (1). Then, it requests a time slot to the SeS for the task (2). The SeS uses the metadata stored in the SMR (3) to infer in which node the task will be executed and informs the CM whether the task has been successfully scheduled or cancelled (4). When the time to execute the task arrives, the SeS contacts with the selected RM to request the creation of a VM to execute the task (5).

When the RM receives the SeS request, it creates a VM and an AM that monitors the SLA fulfillment for this task (6) and notifies the SeS when they are ready (7). The SeS forwards the message to the CM including the access information to the VM. At this point, the CM can submit the task to the assigned VM (8). Once the task is running, it is monitored by the corresponding AM to detect SLA violations (9). If this occurs, the AM requests for more resources to the RM (10), trying to solve the SLA violation locally (11). If the SLA violation cannot be solved locally, the AM informs the CM about this situation (12) who should decide to resubmit the task with higher resource requirements or notify the client if the resubmission fails.

### 4 Semantics in Resource Allocation

#### 4.1 Resource Allocation Ontology

The resource allocation ontology is based on the work presented in [1]. It defines Agents (as Requesters or Providers), Activities, Resources and relations between them to describe how the usage of SP’s resources can be coordinated. The coordination process presents many similarities with the resource allocation. So, classes defined in this ontology can be used in our model. For instance, the CM and RM components can be mapped as Requester and Provider Agents and the different client tasks can be mapped as Atomic Activities. Although the original ontology partially describes our system, it is still uncompleted. Therefore, some changes and extensions are required to make the coordination ontology suitable for resource allocation. Required changes are explained in next paragraphs and the main extensions are depicted in figure 2.

The **Resource** class is extended with different subclasses to describe resources used by SPs. For this initial prototype only a set of resource and properties are defined to validate our semantic resource allocation. We define **HardwareResources** which basically includes **CPU, Memory, Disk** and **Network, SoftwareResources** and **File** which describes files required by different **Tasks**. These resources are contained in **Hosts** which is the resource that a **Tasks** can be assigned. We have also extended the **ClientAgent** (a Requester agent)
which describes the client and the business parameters used for the allocation (now we consider customer priority for the SP, but easily extend to others as rewards, penalties...).

The most important change is the Task description. The original class requires a single instance of Resource. However, this resource instance is unknown because the SeS tries to find the best resources for each client task. Therefore, the requires property in the resource allocation ontology contains a set of TaskRequirements which describes the required resources of a client task. Hardware and Software Requirements are used to look for suitable resources to execute the task while DataRequirements can be used for detecting data dependencies to exploit data locality getting the host which already contains the required files.

### 4.2 Annotation of Semantic Descriptions

The semantic annotation process in SERA includes the search of the relevant metadata to be the semantically described, the creation of semantic models and their registration into a repository (SMR) making these descriptions available for all the components of the system. The usage of semantically enriched metadata should not require additional data management for clients or system administrators, so it should be performed automatically and transparent to the user. In the SERA, this task is performed by the CM and the RM.

### Getting resource descriptions: Resource Manager

As it has been described in previous sections, each host of our system is managed by a RM. It is started during the machine boot time and deploys an agent whose first task is to collect all the information required to fill the semantic description. It is obtained by parsing the data published by a resource monitor such as Ganglia [3]. Then, an RDF [12] model is created with this data using the resource allocation ontology. In addition to the resource properties, the RM description is added to the RDF, including the address to contact with its agent. Once, the RDF model is created, it is registered in the SMR to make it accessible for CMs and the SeS.

### Getting task descriptions: Client Manager

When a new task submission request enters the system, a CM Agent is created to manage the task execution in the system. It gets the task description provided by the client and creates an RDF model (according to our ontology) including TaskRequirements and the client description who request the task. This task model is also created according to our ontology and registered into the SMR. Simultaneously to reading the task description, the client agent uses the task requirements to build a semantic SPARQL [14] query. It is used to select from the SMR all the hosts that fulfill the resource requirements of the task. The query results are inserted in the task model which is registered to the SMR. Performing this step in the CM, the SeS is unloaded of the task-machine matchmaking.

Regarding the client information, we assume that a contract has been previously agreed by client and SP and due to the historical client data the SP administrator will assign a priority for this client, and that all the client data have been also registered in the SMR.

### 4.3 Semantic Scheduler

The SeS is a proof of concept implementation of decision taking using semantics and reasoning with Horn rules, providing an extensible way to schedule tasks (only changing or adding rules). The overview of the SeS is that a rule engine infers a task scheduling into the available resources from the metadata described semantically evaluating a set of rules. Once the SeS receives a scheduling request from the CM, it performs the following tasks:

1. **Get the required semantically enriched metadata**

   The first step to perform is to obtain semantic description of the data involved in the scheduling process. This data is maintained and updated in the SMR following the ontology described previously. So all necessary data for the scheduling process is fetched from this repository with a RDF format to be used in the inference. This required metadata includes all host description, description of task which are currently running or scheduled and their owners description (where the business parameters are described).

2. **Inference**

   Using the retrieved data and the ontology, the SeS creates a Jena 2 [7] Model. Some Jena 2 rules which use a set of auxiliary built-ins (rule extensions written as application code) are also attached to the rule engine and will be fired during the inference process. These rules implement the scheduling policies. For the current prototype we have developed the following rules:

   **Rule 1: First Come First Serve.** This rule is fired when a time slot in a machine for a requested task is found us-
ing the FCFS algorithm, taking into account the hardware and software requirements.

**Rule 2: Task reallocation.** If Rule 1 cannot find a time slot in a machine, Rule 2 tries to move scheduled tasks (not already running) from one machine to another in order to find a time slot for the new task.

**Rule 3: Less priority task(s) cancellation.** If Rule 2 is not able to find a solution, then Rule 3 tries to cancel the tasks with less priority than the new one.

3. Interpretation of Results

After the inference process, Jena 2 gives back a deductions’ graph. It contains changes in the original graph which can be mapped as changes in the state of the system or, in other words, that a task has been scheduled, rescheduled or cancelled. The SeS gets all necessary information about the updated tasks evaluating the deductions’ graph. Afterwards, all the affected data gets updated in the SMR, task cancellations are notified to their CM and the scheduled and rescheduled tasks are inserted in the task queue. At the same time, the SeS agent monitors the execution time of the queued tasks. When the execution time is close, it will contact the RM and CM to execute the task.

5 Experimental Environment and Results

A simple experimental testbed has been deployed to test our prototype and to make some preliminary measurement about the overhead introduced in the semantic scheduling. It consists of three machines: the first one is a Pentium D with two CPUs at 3.2GHz with 2GB of RAM which contains the SeS, CM and SMR. Two additional machines are used as RMs. RM1 is a 3GHz Pentium with 2GB of RAM and RM2 has 4 64-bit Intel Xeon CPUs at 3.0GHz and 6GB of RAM. Both can create VMs with preinstalled software (such as GT4 or Tomcat) on a Debian Linux. Most part of the software is written in Java. Scripts to manage VM creation are written in Bash and some libraries for accessing the Xen [15] monitor are written in C. Ontokit [9] is used as SMR and Jade [6] as agent platform.

Unifying host capacity metrics is a key issue, especially for processors. A machine with two cores at 1.33 GHz has a different CPU capacity than a quad core at 2GHz each one. For this reason, we quantify the CPU capacity as the product of desired percentage and the CPU frequency. Due to both hosts have CPUs whose frequency is around 3GHz, the CPU capacity and task requirements are normalized to this 3GHz. So, the CPU offered by RM1 and RM2 are 400% and 100% respectively and the memory capacity is 6144 and 2048 MB. The experiment focuses on the allocation of CPU, nevertheless, these concepts could be applied to other resources like memory, disk or network bandwidth.

We send a total of eight tasks to the system described in Table 1 where CPU Req. is the required CPU, Dur. is the task duration and Priority is the client priority. Deadlines are specified assuming that the initial time is 00:00.

<table>
<thead>
<tr>
<th>Id</th>
<th>CPU Req</th>
<th>Dur.</th>
<th>Deadline</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>120</td>
<td>M</td>
<td>00:40</td>
<td>HIGH</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
<td>M</td>
<td>00:40</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>M</td>
<td>00:40</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>M</td>
<td>00:40</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>M</td>
<td>00:40</td>
<td>LOW</td>
</tr>
<tr>
<td>6</td>
<td>400</td>
<td>M</td>
<td>00:40</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>M</td>
<td>00:40</td>
<td>LOW</td>
</tr>
<tr>
<td>8</td>
<td>300</td>
<td>M</td>
<td>00:40</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

Table 1. Tasks description

The first 4 tasks are sent to test the SeS when there are enough resources to run the tasks (Rule 1). They are scheduled and sent to RM1 and RM2 depending on their requirements creating a VM to run each task. Next 3 tasks exemplify how the SeS reschedules tasks between different RMs (Rule 2). Task 5 demands more than the available CPU, so it is initially queued into the RM1 (see Table table 2.a). Task 6 must be scheduled right after the current running tasks due to its deadline (SeS does not interrupt running jobs). Besides, it cannot be scheduled on RM2 because it requires 400% of CPU. SeS reschedules Task 5 on RM2 and queue Task 6 in RM1 (Table 2.b). The same reasoning is done for Task 7. SeS reschedules Task 5 after Task 6, and queues Task 7 in RM2. (Table 2.c). The final task will perform a task cancellation (Rule 3). Tasks 5 and 8 have the same deadline and RM2 cannot fulfill their requirements. SeS cancels Task 8 due to its low priority (Table 2.d).

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM1</td>
<td>RM1</td>
<td>RM2</td>
<td>RM2</td>
</tr>
<tr>
<td>TASK5</td>
<td>TASK5</td>
<td>TASK6</td>
<td>TASK6</td>
</tr>
<tr>
<td>TASK5</td>
<td>TASK5</td>
<td>TASK6</td>
<td>TASK6</td>
</tr>
</tbody>
</table>

Table 2. Task rescheduling

The addition of more client tasks and hosts has been simulated in order to test the matchmaking and scheduling processes and to get the expected overhead in production conditions. We have detected that the most important overhead is the time spent in querying the SMR and in the inference process. Other overhead related to the semantic technologies such as the semantic registration is done by several components in parallel and it is neglectable compared with the mentioned above. Figure 3 shows the time spent on querying the SMR to get the host which match with the task requirements. As we can see in the figure, the first queries spend more time to get the results than the other ones. It is due to some initialization issues performed by the Ontokit. We also can see that the larger the number of hosts in the system is, the bigger the variations between the different queries are. On the other hand, the number of tasks in the system does not affect the query time which is the expected behaviour because these queries only look for hosts
that match with one task. Regarding the inference time it is depicted Figure 4. It includes the creation of the data model and inference with the rule engine. Two different zones can be distinguished in the figure. The first one is when the resources are not full and only the Rule1 is fired to perform the scheduling. The second zone is when Rule1 is not fired and the rules 2 and 3 must perform the scheduling. This zone produces a faster increment of the scheduling time.

6 Conclusions and Future Work

This paper introduces a working framework for facilitating service provider management, which allows reducing costs and at the same time fulfilling the quality of service agreed with the customers. Our solution exploits the features of semantic technologies to perform the task scheduling taking into account task requirements as well as business parameters, the virtualization for building on-demand execution environments, and agents to react in case of failures. We have focused in the part of the treatment of the semantically-enriched metadata in order to assign SP resources to client task in the way that the best clients are favored. We have proposed some extension to an existing resource ontology to fulfill our needs as well as a small set of inference rules to guide the inference process.

The behavior of the semantic scheduler is the expected, offering flexibility to re-schedule and cancel tasks to meet requests with more priority. The overheads of the matchmaking queries are significant at the initialization phase, but remain reasonable on the steady state. While the inference time grows linearly with the number of tasks, we consider this time still reasonable if we consider the granularity of the requested tasks. As future work, we consider the extension of the ontology and the set of rules that will take into account further business parameters. Additionally, we will study how inference overheads can be reduced by minimizing the amount of data considered by the rule engine applying previous filters.

7 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 the European Commission under FP6 IST contract 034556 (BREIN).

References