Parallel Processing Letters
© World Scientific Publishing Company

SCHEDULING AND MANAGEMENT OF VIRTUAL RESOURCES IN GRID SITES: THE SITE RESOURCE SCHEDULER[*]

RODRIGO N. CALHEIROS
Faculty of Informatics, Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, Rio Grande do Sul 90619-900, Brazil

TIAGO FERRETO
Faculty of Informatics, Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, Rio Grande do Sul 90619-900, Brazil

and

CÉSAR A. F. DE ROSE[†]
Faculty of Informatics, Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, Rio Grande do Sul 90619-900, Brazil

Received December 2006
Revised March 2008
Communicated by A. Bode

ABSTRACT
This paper presents a new approach to resource management and scheduling in computational grids, in order to simplify the vision users have from grid resources and their management. Scheduling decisions are moved to the site where resources are hosted, allowing quick response to changes in load and resource availability. Users do not need to be aware of the resources they use and are instead supplied with "virtual resources" representing the amount of computational power available to them in the site. This approach results in new challenges, like the management of two-level scheduling schema and the need to define a site capacity measure, but simplifies and optimizes scheduling in grids. In this paper we present details of this approach – called Site Resource Scheduler (SRS) – as well as some issues regarding its simulated performance and its deployment in a site. We show that the main advantage of this approach is an overall reduction in the execution time of tasks in most scenarios.

Keywords: Grid Computing, Resource Management, Resource Scheduling

1. Introduction

Grid computing has become an important tool for both science and business. The main goal of grid computing is to provide easy access to resources spread around the world, under control of several organizations [14]. Despite the constant increase

[*]This work was developed in collaboration with HP Brazil R&D.
[†]Corresponding author. E-mail address: cesar.derose@pucrs.br

of worldwide grid usage, some areas, such as resource management and resource scheduling still need major advances. Resource manager responsibilities are discovering, dealing and monitoring grid resources [9], while the resource scheduler is responsible for mapping user tasks to resources that are not under direct control of the user [19]. Nowadays, users are responsible, by themselves or through grid brokers, to discover, allocate and manage grid resources. When a user needs to access hundreds of resources spread around the world, management can be inefficient since users usually do not know accurate details of resources. Scalability issues can also emerge. Such problems are more critical when the resources available to users are non-dedicated, i.e., resources can leave or join the grid at any time. In this case, monitoring of resource activity and rescheduling is necessary.

Clusters of workstations were once the resource preferred by scientists to execute their experiments with high processing power demands. However, with the advent of more complex problems, using a single cluster was not affordable. In order to handle these problems, the utilization of several clusters together was the most reasonable step to take, letting to the realization of the first grids (e.g. I-WAY [11]).

One key issue in using clusters in computational grids is that, in general, grid users must provide the amount of time and nodes they want to allocate to the Cluster Resource Manager (CRM). Nevertheless, grid users would rather allocate the maximum computer resources and time they can in order to complete their tasks as soon as possible, which means they usually do not desire to specify allocation constraints. Moreover, if the grid is planetary-scale, users do not know the characteristics of all clusters in the grid, making it impossible for them to estimate task duration.

In order to overcome such limitation, in [18] an alternative approach to use clusters in grids is presented. This strategy is based on opportunistic computing techniques and does not make use of a formal allocation request to cluster resource managers. It facilitates the utilization of resources in grids. Nevertheless, as cluster resources supplied in that way are non-dedicated to the grid, at any moment they can be removed from the grid, aborting any task being executed in these resources.

This approach increases considerably the number of resource faults, since for the user each resource removed from the grid is a faulty resource. However, it can be enhanced with the inclusion of another scheduling level in the site, reducing the number of faults that must be handled by the user and, therefore, minimizing the impact of faults in the performance of grid applications.

In this paper we present the Site Resource Scheduler (SRS). The goal of the SRS is to simplify the vision users have of grid resources and their management, moving scheduling decisions from the grid scheduler to the site where resources are hosted, allowing quick response to changes in both load and resource availability. This vision is reached through an extra resource management layer in grid computing (two-layer scheduling), that hides resources specific issues (resource virtualization) and enables system administrators to have a better control over resource utilization. With SRS, users see a site as a “powerful” virtual resource instead of a set of real machines.

The actual power of this virtual resource varies according to site load and user access rights. Experiments performed with a SRS prototype using the OurGrid middleware [7] show that SRS can reduce the execution time of grid applications, especially when large tasks have to be scheduled in non-dedicated resources. In general, by virtualizing all site resources in one machine, the number of resources to be managed is reduced to the number of sites an application knows, simplifying the scheduling operation considerably.

The remaining of this paper is organized as follows: Section 2 presents some works related to our proposal, and shows how our approach differs from the existing ones. Section 3 presents the Site Resource Scheduler, its architecture, challenges emerging from the approach and how it might be overcome. Section 4 presents some preliminary results obtained in simulation of the system. Section 5 presents a case study, a deployment of SRS using the OurGrid middleware. This section also presents some experiments performed with the prototype, their results and some issues about SRS deployment in Globus grids. Concluding remarks and further work are discussed in Section 6.

2. Related Work

GRAM (Grid Resource and Allocation Management) is currently a well-known solution for resource management in grids, due to the widespread utilization of Globus Toolkit [12] to build grids. The operation of GRAM is quite simple: each resource, which can be a single machine or a collection of machines already managed by another resource manager (e.g. OpenPBS [6] or Condor pool [16]), located in a site executes a GRAM module instance. The GRAM module converts requests from the grid to the appropriate resource manager. Our proposal differentiates itself from GRAM since we use a single module to manage all resources of a site (e.g., a cluster, a set of desktop machines), while using GRAM, a different instance of GRAM needs to be present for each resource available to the grid in the same site.

Condor [21] is a system originally proposed as a solution for exploitation of idle resources in local area networks. However, it has evolved into a system to manage resources in cluster and grids. It implements a idleness detector and have mechanisms to provide application checkpointing, among other functionalities. In spite of being a system to exploit idle computing resources, deploying Condor in a complex environment, composed by desktop machines (to be used when idle) and clusters, requires replacement of the CRM of each cluster by Condor. In such case, clusters will be also managed by Condor. We believe this approach is not acceptable in most cases, as administrators want to keep their systems as is to agree in donate them to the grid. Our strategy, on the other hand, allows deployment of any type of local resources without changes in any software of the system. It is only necessary to install SRS in the machine that acts as entry to the grid in order to allow usage of not only desktops, but also resources managed by different clusters schedulers in the grid.

Virtual Grid [15,24] is an approach to deliver virtual resources to grid users. It provides a resource description language and a resource discoverer. However, in this

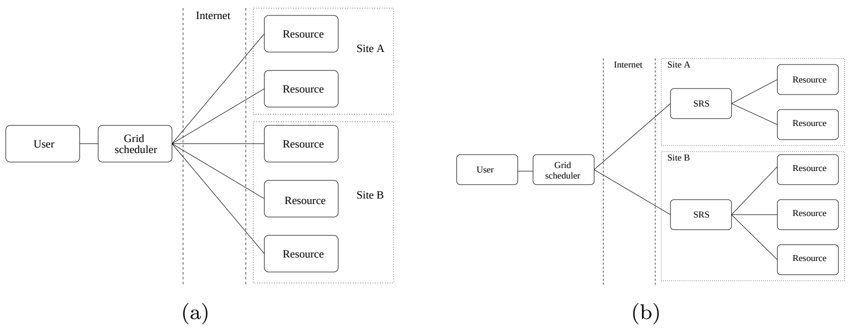

Fig. 1. Grid resources access (a) without SRS (b) with SRS.

approach the virtualization happens in the user side, below user application. Our strategy, on the other hand, virtualizes resources at the provider side, delivering to the user system virtualized resources.

OurGrid *middleware*[‡][7] is a complete infrastructure for deploying computational grids focusing in BoT (Bag-of-Tasks) applications and is used in the Brazilian grid [10]. OurGrid provides, as its main components for resource access and scheduling, two modules called Peer and MyGrid. Peer is responsible for connecting to other Peers and discovering resources to users (through the MyGrid broker and scheduler). Each user can be connected to only one Peer. The Peer basically controls resource distribution among other Peers and local users. MyGrid is used by each user to schedule tasks using all available resources provided by its Peer. The difference between OurGrid and SRS is that the former delegate real resources to grid users, and the users must manage and schedule their tasks. If the resource becomes unavailable to the user, the task must be rescheduled, which requires resubmission of input files. Our approach hides resources unavailabilities from the grid users. Thus, the resubmission of files and the rescheduling are not necessary.

Depending on the grid middleware to be supported by the site, SRS will be deployed as a add-on on the grid software (e.g., GRAM, OurGrid's Peer). However, SRS could be implemented as an independent middleware. In such case, no other software would be necessary to support grid computing.

3. Site Resource Scheduler

On current grid middleware [13,7], regular tasks such as resource discovering, allocation and management are delegated to users, directly or through grid schedulers (Figure 1 (a)). When a user has access to thousands or even hundreds of machines spread around the world, these tasks can be hardly accomplished in an efficient manner.

[‡]www.ourgrid.org

This issue is even more critical using non-dedicated resources. In this case there is an extended overhead due to the need to monitor resource's activity, and to reschedule tasks to other machines if needed. Rescheduling usually accompanies a resubmission of input files, which delays task execution even more.

In this paper, we refer to the process of losing resources as a situation where a non-dedicated resource is requested to be used again by its regular local user. By *local users*, we mean users that have an account in the site to directly access the cluster management system (e.g., PBS). In opposite, we also consider in this paper the existence of *grid users* that are users that access resources from several sites (being possible that some of these resources were cluster resources) around the world, directly or through some grid scheduler, and want to run grid applications on them.

The lost of resources can happen in different scenarios. It can happen because the grid is stealing cycles from a machine that is not being fully used (e.g. public computing [4] projects) or because the site configuration, in order to prioritize local users, allows preemption of grid tasks being executed when a local demand arises, ensuring that the grid will not delay local users' applications. Yet, if the resource is part of a cluster, it can be delivered to the grid when idle, due to scheduler queue fragmentation, using the technique presented in [18], and returns to the cluster when a regular cluster request is performed by a local user.

The motivation behind the Site Resource Scheduler (SRS) is to simplify and optimize the utilization of site resources by grid users, and also enhance the management of these resources regulated by site policies. SRS provides a virtual representation of available site resources (resource virtualization) to the grid, which are used by grid users as real resources (Figure 1 (b)). Another goal of SRS is to minimize scheduling and resource management overheads performing site-level scheduling, i.e., tasks submitted to virtual resources are internally scheduled to real machines, and internal site faults, due to resources lost, are handled transparently by SRS whenever suitable (e.g. BoT job, parameter sweep job).

Resource virtualization is achieved using an abstraction of all computational resources presented in the site. Users requesting resources do not receive real machines from the site. Instead, an amount of "virtual resources" is supplied to users. The goal of such approach is to hide resources unavailabilities, avoiding both resubmissions and rescheduling of tasks. This approach introduces a new way of representing the site capacity in the interaction with regular grid schedulers.

3.1. Resource Virtualization

In order to provide the virtualization level proposed, real machines should not be directly delegated to users. Instead, an abstract measure, describing the current site capacity available to that, should be given. The same approach is used by users: their requirements must be described as an abstract measure, being the site responsible to translate it to a real amount of resources. Another important issue related to resource usage is that resources should not be monopolized by a single user: in order to improve overall grid performance, the site must serve as many users as possible, without harming local users.

Using SRS, the exposure of site resources to the grid shifts from a set of real resources to a metric describing site's capacity. Such capacity varies in function of a site's current load, i.e., the amount of resources that is in use in a given moment, and based on the access rights of the grid user requesting sites' capacity.

The usefulness of deploying an abstract measure of a site's capacity, as well what kind of abstraction representation should be used depends on how the grid scheduler maps tasks to resources.

One possible measure of site's capacity delivered to grid users is the *amount of virtual resources* available to them. Using this approach, a user receives a number of virtual resources and deals with them as if they were real resources. However, each virtual resource can represent zero, one or multiple actual resources: it will depend on SRS internal decisions and will not be informed to the user an may vary in time. A virtual resource mapped to zero resources means that tasks submitted by the user to that resource will be queued in the scheduler's queue and will be executed when possible.

A virtual resource mapped to one resource will sequentially forward tasks directly to the resource on where this virtual resource is mapped. If more than one task were submitted at once, the first one will be executed and the other will be queued. A mapping from a virtual resource to multiple resources means that if more than one task were submitted through that virtual resource, each task will simultaneously be submitted to a different physical resource (up to the limit of mapped machines) and the tasks exceeding the amount of virtual machines will be queued to a lately execution. It is worth noting that, using this approach, users are not aware on the actual mapping in a given time between virtual and actual resources.

From the grid user perspective the (virtual) resources allocated to them are always available: changes in the internal mapping between virtual and actual resources are kept hidden from the users. Using this strategy, it is possible to avoid a new data transfer and application submission, since the data is already stored in the site and the task execution will be restarted when a new resource becomes available to the user.

This approach is not useful if the grid scheduler is able only to sequentially assign tasks to resources (e.g. due to application characteristics), i.e., a second task will be assigned to a resource only after the first one finishes. In such a case, a mapping from a virtual resource to multiple actual resources is not efficient: if the virtual resource is mapped to two real resources, the second one will be idle, waiting for another task from the user, and this task will not arrive until the completion of the first. Thus, it is important that grid schedulers could send more tasks than the number of virtual resources available in order to efficiently use the resource virtualization feature provided by SRS.

When the grid scheduler is able to distribute tasks proportionally to the capacity of each available resource, a more abstract measure can be used. For example, the method proposed in [23] uses benchmarks to determine resources capacity. We suggest a site's capacity measurement that also uses relative performance, but using several benchmarks, each one evaluating some aspect from the machine: performance on floating point operations (e.g. CFP2000, from the benchmark suite SPEC CPU2000 [1]), performance on integer operations (e.g. CINT2000 from the same suite) and performance on I/O.

It is worth noting that there are other requirements to be considered by users when describing job requirements. It is necessary not only to specify the capacity required but also all specific requirements of the job. Job requirements include: specific operating system or architecture, minimum amount of RAM, or specific software running in the site (e.g. a Data Base Management System). Before returning site's capacity to users, these specific requirements should be considered, with other site specific requirements, such as user access rights, current site load, advanced reservation, and future load expectation.

3.2. Architecture

The SRS architecture is presented in Figure 2 and is composed by the following modules:

- The *Information module* stores information about site resources and users. This information is useful to improve scheduling performance and to enforce local policies. Information about resources includes resource state (available or not available to the grid), operating system, architecture, CPU rate and RAM memory. Information about users includes user access rights and historic information about usage of resources of a site;

- The *User Interface module* receives user requests and forwards it to the appropriate module. Services available to grid users include those related to resource management (e.g., determination of site availability, dedicated and non-dedicated allocation and resources reservation), job management (e.g., job execution, monitoring and cancellation), file management (e.g., temporary file storage, persistent file storage, files listing and exclusion, file download), security and access control (e.g., user identification), and user accounting and balance;

- The *Resource Management module* controls all site resources, assigning those available to the grid to the Scheduling module and feeding the Information module with information about resources utilization;

- The *Scheduling module* receives information about tasks submitted through the User Interface module, information received from Resource Management module about available resources and performs task mapping and execution.

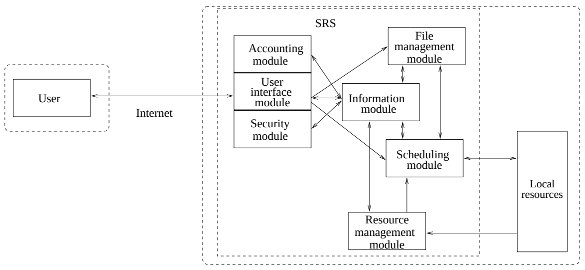

Fig. 2. SRS architecture.

This module also interacts with local resource managers in order to get access to space-shared resources;

- The *File Management module* manages staged files. When files are requested by user tasks, they are transferred to the local resource executing the task. It also retrieves tasks results from the resources and stores them;
- The *Security module* is responsible to apply security protocols, defined by the site administrators;
- The *Accounting module* accounts resource utilization and applies accounting policies, if necessary, to resources.

Because of the high flexibility of the SRS architecture, the utilization of all modules is not required. SRS can be used in combination with other grid middleware, where middleware and SRS modules can be mixed in order to include extra functionalities desired by the site administrator.

3.3. Job Execution

In this section we describe the steps to execute jobs in a site using SRS. The first step is to query SRS to obtain current site's capacity available to the grid user. The query includes client's identification (e.g. credential) and a description of user job requirements. This information is necessary in order to return the correct site's capacity according to user access rights and job requirements.

After receiving information about site's capacity, a user can make a request for an amount of the overall site's capacity. After the reception of requests confirmation, jobs are submitted to SRS. Each job submission is executed using a job specification which contains information such as: files being staged, storage method (temporary or persistent), application name, procedure to handle output files, etc. For each job submission, SRS returns a job identifier (job ID), which is used to interact with the job through SRS (e.g., start execution, monitor activity, stop execution, retrieve output files, etc).

Internally, SRS aggregates resources obtained from different places, for example, a cluster managed by PBS [6], a cluster managed by Crono [17], or a set of machines managed by a Condor [16] pool. Grid tasks are sent to these resources and executed until the resource be requested by a local user. A grid task canceled due a local request returns to the SRS execution queue to be mapped to another available resource, and this action are not communicated to the grid user: queries from the grid user about the execution status will return "running" even if the tasks were in the scheduler queue. If the job failure are not caused by preemption nor by physical resource failure, the task will be returned to the user as "failed".

The steps required by SRS to job submission and management are somehow provided by existing protocols (e.g. OurGrid's protocol for resource description and job management, GRAM's JDF and JSDL [5] to resource description, and WSRF [8] to job management). Thus, applying SRS with existing grid middleware will not require changes in the applications. Instead, only a new module on each site is enough to provide SRS functionalities, as presented in the Section 2.

It is worth noting that resources used by SRS can be non-dedicated (like Condor resources and cluster resources with a transparent allocation approach [18]) or dedicated. In the later case, SRS itself allocates resources from cluster managers and use them to execute grid tasks. The decision of which approach to use is delegated to the site administrator. We believe that allowing administrators to donate dedicated resources in a non-dedicated way can incentive them to donate resources to the grid due to the minimal interference caused to local users.

4. Simulation Experiments

In order to evaluate SRS in a large scale grid environment, a set of simulations was performed with GridSim Toolkit v. 3.3 [20]. The goal of these experiments is to show how SRS impacts on application's execution time. To reach such goal, the environment built for the simulation is based in a subset of the World Wide Grid (WWG) [2] environment. WWG is a grid testbed encompassing 63 sites representing 21 countries from Europe, South and North America, Asia, and Oceania. By August 2006, there were over than 3000 resources in WWG.

Ten WWG sites compose the simulation environment, and are presented in Table 1. Each site contains a cluster of workstations composed by Pentium machines. The table also presents for each site: name, location, processor type, and amount of available machines. In the simulation all sites use SRS to manage their resources. The simulation environment has a total of 325 machines.

GridSim requires the specification of resources capacity in Million Instructions Per Second (MIPS). However, such information about WWG resources is not available. Thus, we defined two values for MIPS of resources, depending on the CPU type: 344 MIPS and 684 MIPS, typical values for a Pentium 3 running at 800MHz and a Pentium 4 at 2GHz, respectively, according the SPEC benchmark [1].

The load is simulated as follows. Each machine has a probability equal to the

Table 1. Subset of the WWG used in the simulations.

Site name	Location	Processor type	Machines
amata1.cpe.ku.ac.th	Thailand	Pentium 3	16
aurora.cs.usm.my	Malaysia	Pentium 3	18
carcara.lncc.br	Brazil	Pentium 3	32
galley.doshisha.ac.jp	Japan	Pentium 3	16
gideon.csis.hku.hk	China	Pentium 3	32
hathor.csse.manash.edu.au	Australia	Pentium 3	64
leo.cs.wcu.edu	USA	Pentium 3	16
mercury.sao.nrc.ca	Canada	Pentium 4	50
pacifica.iridis.soton.ac.uk	UK	Pentium 4	16
venus.gridcenter.or.kr	Korea	Pentium 4	65

site's load (a simulation parameter) of being occupied at simulation start. In the tests presented in this paper, the load was 80%. Thus, each machine will have a probability of 80% of starting occupied. Afterwards, a periodic test is performed in order to decide if more resources will be removed from the grid. In order to prevent unavailability of machines in a site, even under high load, each resource returns after 5 minutes to the grid.

The grid workload is composed by grid tasks from 5 different users. Each user submitted 50 tasks to the grid. Each task requires the transfer of input files and sends back output results. The file size applied for these files are 100kB, 1MB e 10MB (typical values from real applications submitted in the OurGrid grid). Execution time of each task has been set to 600 seconds in the first experiment and to 900 seconds in the second experiment (in both cases, with a random variation of 10% for each task). These times are related to the execution in the Pentium 3 machines.

In the simulation one virtual resource is created for each machine available to the grid in the site. Two ways of deciding the amount of virtual resources supplied to a user were tested. In the first one, virtual resources are supplied without virtual resources redistribution. Thus, a user requesting all the machines will get them, and further request from other users will not be attended. In the second approach, virtual resources are redistributed among grid users when new requests arrive, in such way that each user receives approximately the same amount of machines.

Complementary to the two ways of distribution of virtual resources, scheduling is done either by FCFS or by an adaptive approach. The former is a simple first-come-first-served (FCFS) approach: tasks wait in the queue and requests are served in the same order they arrive in the queue. In the later, it was adopted a priority policy where tasks from the user with the bigger difference between number of virtual resources requested by him and number of tasks running in actual resources have preference in the scheduling.

The combination of the two schemes of virtual resource distribution and two schemes of mapping generated four SRS simulation scenarios, as showed in Table 2.

Table 2: SRS simulation scenarios.

		Mapping policy	
		FCFS	priority-based
Resource redistribution	no	SRS1	SRS2
	yes	SRS3	SRS4

The simulation includes a simple non-dedicated scheme without SRS, where tasks not completed due to the removal of the resource from the grid (in behalf of a local user) are rescheduled by the grid user (through the grid scheduler). Each scenario was executed 30 times, and the average completion time of all tasks from all users was measured. The values obtained were normalized in function of values obtained for the non-dedicated policy. This way, each value obtained represents the relative performance of the policy in relation of the performance of the non-dedicated policy.

Results presented in this section do not represent a comparison between WWG performance with and without SRS at all: it only represents performance of SRS against a non-dedicated policy of scheduling resources to grid users. The non-dedication scenario, which is used as the reference time to evaluate SRS, is not the default policy applied in the WWG's sites. We have took advantage only of the description of physical resources of WWG, and not of the policy used on each site.

Figures 3 and 4 show, respectively, results from the experiments composed by tasks with 600 and 900 seconds. The unit used in the figures is the average execution time of each approach, normalized with the time obtained to the non-dedicated policy. From the analysis of results from Figure 3 (600 seconds), we see that both SRS1 and SRS2 introduced performance degradation for all file sizes. This overhead varied from 1% (SRS2 for 1MB) up to 23% (SRS1, 1MB). On the other side, SRS3 and SRS4 presented a performance increase, reducing the overall execution time up by 27% (SRS4, 1kB), due to the resource redistribution policy, which benefits all the users, helping to reduce execution time of their jobs. Without redistribution, users that request resources earlier than the others will get more resources and improve their single performance, but the other users will have their performance harmed, increasing the overall execution time.

Increasing tasks size to 900 minutes, as presented in Figure 4, makes the performance of SRS1 only decrease when transferring 1 kB files. In the other 2 cases, a small enhance in the performance was noticed. Simulations showed increase in performance for most of the cases, in which the best result was 67% (SRS4 with 1kB file transfer).

However, in the case where 10MB of data are transferred, all the scenarios presented similar performance, with a performance loss in SRS4 of 4% (this is the only case where SRS4 presents loss of performance). SRS3 is the only policy that presented a performance increase for all cases, while SRS2 only enhanced performance transferring 10MB files in tasks of 900 seconds.

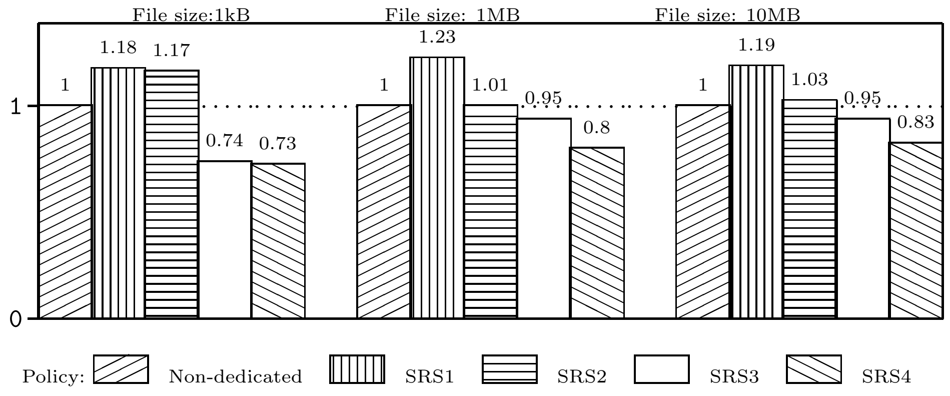

Figure 3: Normalized average execution time of 600-seconds tasks.

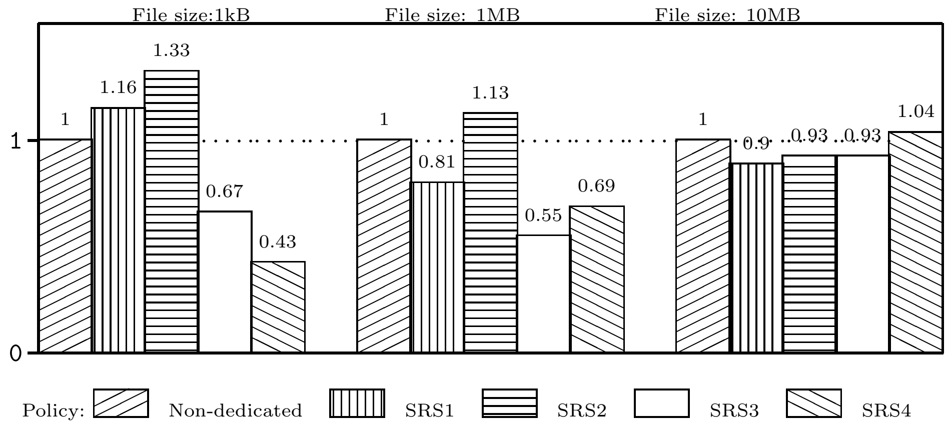

Figure 4: Normalized average execution time of 900-seconds tasks.

Based on the results presented we can conclude that a policy with resources redistribution may enhance overall grid performance. Regarding the scheduling scheme, both FCFS and priority-based scheduling can increase system performance depending on the workload.

5. Case Study

In order to validate our proposal, we developed a SRS prototype for the OurGrid middleware used in a Brazilian grid composed by 15 sites with more than 300 machines. Because of the open architecture of OurGrid and its ability to link additional modules, we implemented the SRS module using the same communication protocol used in OurGrid to maintain its compatibility with the original implementation, and were able to perform tests with an already deployed infrastructure.

In OurGrid, a resource, called GuM, is defined according to correspondent machine properties, because there is a 1:1 relation between a GuM and the resource it represents. Although OurGrid's high configurability, using its original protocol reduced the possibilities to represent a site's capacity, as MyGrid (OurGrid's grid scheduler and resource broker) can only manage, monitor, and submit tasks to GuMs.

To keep compatibility between OurGrid and SRS, we adopted the utilization of a metric representing the number of virtual resources available to grid users, called SRSGuMs. The number of SRSGuMs in a given time is the same number of processors (in opposite of the number of computers) available in the site. SRSGuMs receive properties of a machine that has originated it. For instance, if a machine is a dual processor Itanium 2 with 1024MB of RAM, SRS will generate 2 SRSGuMs, each one described as a single processor Itanium 2 with 1024MB and having all the other properties from the real machine.

These virtual resources are generated to give an estimate of computing capacity of the site. It is possible that in a moment there will not be any machine available to be mapped to a SRSGuM. In this situation, a task submitted to the SRSGuM keeps waiting in the scheduler queue until one machine becomes available, as previously described.

SRSGuMs are made available to clients as they request resources. Although OurGrid maps a GuM directly to a physical resource and every request is forwarded to that machine, using SRS those requests are forwarded to a virtual resource in the machine running SRS, and this resource forwards the request to the module able to execute the job. If the service is related to task execution, the request is forwarded to the *Scheduling module*. If the service is related to file management, the request is forward to the *File Management module*. Users are not aware that they are not accessing a real resource. From their point of view, all requests are forwarded to real machines.

5.1. Evaluation

To analyze the impact of management performed by SRS using OurGrid, some experiments have been performed using two OurGrid sites located 4000 Km apart. One site has been used as a resource consumer (LSD/Campina Grande), and the other as a supplier (CPPH/Porto Alegre), hosting a cluster whose machines have been delivered opportunistically to the grid. The cluster is composed of 48 machines and 67 processors: 11 Pentium 3 1GHz with 256MB of memory, 10 Pentium 4 1.6GHz with 256MB of memory, 9 Dual Pentium 3 550MHz with 256MB of memory, 4 Dual Pentium 3 1GHz with 256MB of memory, 8 Pentium 4 2.8GHz with 2.5GB of memory, and 6 Dual Xeon 3.6GHz with 2GB of memory.

The test comprised the execution of 12 tasks, each one with execution duration of 5 minutes (controlled by a sleep call). Each task was assigned to a machine, and at any moment a machine can run only one task. The experiment was executed using the same file sizes used in the simulation: 100kB, 1MB, and 10MB (this is

Table 3. Results for 12 tasks of 5 minutes.

File size	OurGrid	OurGrid + SRS	SRS Overhead
0	312s	313s	0.32%
100kB	396s	398s	0.51%
1MB	1267s	1283s	1.26%
10MB	18530s	10100s	(performance increase of 45.5%)

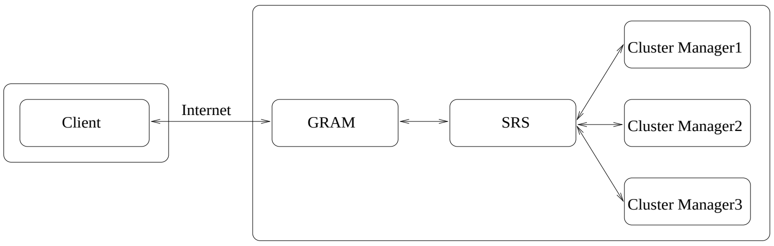

Fig. 5. SRS as a module between GRAM and site resources.

the amount of information to be transferred for each execution). To simulate a non-dedicated execution, resources were randomly removed from the grid every 10 minutes. Table 3 compares the results obtained in the OurGrid middleware with and without SRS. Each line presents results for a specific amount of file size.

As expected, we can see that using SRS significantly reduces the execution time of jobs that perform larger file transfers. This is usually the case in sites that provide non-dedicated access to its resources, were tasks are more often rescheduled. In other types of sites, we observe that SRS can be used with low performance impact to grid users. When a job requires a large amount of data, staging files in SRS before transferring to the real resource provides an efficient reduction of job execution time. When tasks do not require file transfers, both approaches present an equivalent performance, as showed in the first line of Table 3.

Observing the results obtained we can conclude that the utilization of SRS caused a little overhead for small files. Application execution time increased when machines do not fail or files are too small to cause an impact in resubmission. This happens due to the inclusion of the additional scheduling layer. We consider this a tolerable overhead since in most cases grid tasks will need to be rescheduled due to the dynamicity of resources in the grid, resulting in a higher potential performance gain using SRS.

5.2. SRS on Globus grids

Deployment of SRS in a Globus grid is another issue being considered in our research. In the proposed approach, SRS would be deployed as a layer between GRAM and all site resources, as presented in Figure 5. This approach allows using a single GRAM to manage all site resources, instead of a GRAM for each site resource. It happens due to GRAM's point of view, there is only one resource to be accessed, the one managed by SRS. Resources delivered to users are virtual ones, in the same way it has been done in the OurGrid case study.

It is necessary, however, to supply a GRAM Adapter able to interact with SRS, which should be able to interact with all site resources. With this model, there is no change in the GRAM Front end, thus there is no need to change grid schedulers. File management and information services do not need to be implemented because it is possible to use GridFTP [3] and MDS [22] to perform these tasks. It is worth noting that Globus *multijobs* are the most appropriate kind of application to be used, as this kind of job can be easily fractionated and distributed among real resources, after being submitted to a virtual resource by the user.

6. Conclusion and Further Work

As the number of resources available in grids increases, scalability of grid schedulers becomes a relevant issue to grid computing. The management of hundreds (even thousands) of heterogeneous resources spread around the world may have a huge impact on the performance of task scheduling. In this context, we presented in this paper an approach to manage and schedule grid resources called Site Resource Scheduler (SRS). It virtualizes not only all heterogeneous and possibly non-dedicated resources but also clusters running different schedulers of a site into a powerful machine, simplifying management and scheduling in grids.

We showed through empirical experiments on a real grid and also with simulation that the main advantage of this approach is an overall reduction in the execution time of tasks (up to 57%) in most scenarios. This improvement is a direct consequence of the proposed two-level scheduling that is able to better react to non-dedicated resources. This scheduling scheme is also more scalable and increases the fairness when accessing site resources. There are still some issues that need to be addressed. Since the utilization of SRS does not imply a specific scheduling heuristic, new scheduling heuristics could be developed to profit from the resulting two-level scheduling schema. We are also simulating SRS performance in other environments in order to best evaluate its effectiveness.

References

[1] Standard performance evaluation corporation (spec). http://www.spec.org.

[2] World wide grid. http://www.gridbus.org/ecogrid/wwg/.

[3] W. Allcock. Gridftp: Protocol extensions to ftp for the grid. Global Grid Forum Recommendation GFD.20. Available at http://www.ggf.org/documents/GFD.20.pdf.

[4] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, 2004.

[5] A. Anjomshoaa et al. Job submission description language (jsdl) specification v1.0. Available at http://www.gridforum.org/documents/GFD.56.pdf.

[6] A. Bayucan. Portable batch system administration guide, 2000.

[7] W. Cirne et al. Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006.

[8] K. Czajkowski et al. The WS-resource framework version 1.0. Available at http://www.globus.org/wsrf/specs/ws-wsrf.pdf.

[9] K. Czajkowski, I. Foster, and C. Kesselman. Agreement-based resource management. *Proceedings of the IEEE*, 93(3):631–643, 2005.

[10] C. A. F. De Rose et al. GerpavGrid: using the grid to maintain the city road system. In *18th International Symposium on Computer Architecture and High Performance Computing*, 2006.

[11] T. A. DeFanti et al. Overview of the I-WAY: Wide-area visual supercomputing. *International Journal of Supercomputer Applications and High Performance Computing*, 10(2/3):123–131, 1996.

[12] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network & Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[13] I. Foster and C. Kesselman. The Globus toolkit. In *The Grid: Blueprint for a New Computing Infrastructure*, chapter 11, pages 259–278. Morgan Kaufmann Publishers, 1999.

[14] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *The International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[15] Y. Kee et al. Efficient resource description and high quality selection for virtual grids. In *IEEE International Symposium on Cluster Computing and the Grid*, 2005.

[16] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor – a hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, 1988.

[17] M. A. S. Netto and C. A. De Rose. CRONO: A configurable and easy to maintain resource manager optimized for small and mid-size GNU/Linux cluster. In *Proceedings of the 2003 International Conference on Parallel Processing*, 2003.

[18] M. A. S. Netto et al. Transparent resource allocation to exploit idle cluster nodes for execution of independent grid tasks. In *Proceedings of the 1st International Conference on e-Science and Grid Computing*, 2005.

[19] J. M. Schopf. Ten actions when grid scheduling. In J. Nabrzyski, J. M. Schopf, and J. Węglarz, editors, *Grid Resource Management: State of the Art and Future Trends*, chapter 2, pages 15–23. Kluwer Academic Publishers, Norwell, 2003.

[20] A. Sulistio, G. Poduval, R. Buyya, and C.-K. Tham. Constructing a grid simulation with differentiated network service using GridSim. In *Proceedings of the 6th International Conference on Internet Computing*, 2005.

[21] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In *Grid Computing: Making the Global Infrastructure a Reality*, chapter 11, pages 299–336. John Wiley and Sons, West Sussex, 2003.

[22] G. von Laszewski et al. A directory service for configuring high-performance distributed computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, 1997.

[23] L. Xiao, Y. Zhu, L. M. Ni, and Z. Xu. GridIS: an incentive-based grid scheduling. In *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium*, 2005.

[24] Y. Zhang et al. Scalable grid application scheduling via decoupled resource selection and scheduling. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, 2006.