CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2012)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.2832

Implementing and evaluating scheduling policies in gLite middleware

A. Kretsis [1], P. Kokkinos [1,*,†] and E. A. Varvarigos [2]

[1] *Computer Engineering and Informatics Department, University of Patras, Greece*
[2] *Research Academic Computer and Technology Institute, Patras, Greece*

SUMMARY

Grid scheduling algorithms are usually implemented in a simulation environment using tools that hide the complexity of the Grid and assumptions that are not always realistic. In our work, we describe the steps followed, the difficulties encountered and the solutions provided to develop and evaluate a scheduling policy, initially implemented in a simulation environment, in the gLite Grid middleware. Our focus is on a scheduling algorithm that allocates in a fair way the available resources among the requested users or jobs. During the actual implementation of this algorithm in gLite, we observed that the validity of the information used by the scheduler for its decisions affects greatly its performance. To improve the accuracy of this information, we developed an internal feedback mechanism that operates along with the scheduling algorithm. Also, a Grid computation resource cannot be shared concurrently between different users or jobs, making it difficult to provide actual fairness. For this reason we investigated the use of virtualization technology in the gLite middleware. We did a proof-of-concept implementation and performed an experimental evaluation of our scheduling algorithm in a small gLite testbed that proves the validity and applicability of our solutions.

Received 11 May 2009; Revised 2 February 2012; Accepted 11 February 2012

KEY WORDS: grid networks; scheduling algorithms; glite middleware

1. INTRODUCTION

The emergence of high speed networks is making the vision of Grids a reality. A number of applications in science, engineering, and commerce can benefit from the use of Grid networks. Grids consist of geographically distributed and heterogeneous computational and storage resources that may belong to different administrative domains, but can be shared among users by establishing a global resource management architecture, called Grid middleware. A Grid middleware is a software package providing a number of fundamental Grid services, such as resource discovery and monitoring, job, data, and resource management. An important issue in Grids is the management of the resources and the scheduling of the jobs. Grids are quite dynamic, with resource availability and load varying rapidly with time, while user jobs have very different characteristics and requirements. The extent to which resource management and scheduling are able to cope with this unpredictable environment is key to the success of Grid networks, because it determines the efficiency in the use of the resources and the QoS provided to the users.

Until now most of the scheduling algorithms proposed for Grids are implemented and evaluated in simulation environments. In general, simulation environments are acceptable scientific tools as can be confirmed by the great majority of scientific papers in Grid and distributed computing related conferences and journals, and are justifiable for a variety of reasons. For example, the evaluation of

*Correspondence to: P. Kokkinos, Computer Engineering and Informatics, University of Patras, Greece.
[†]E-mail: kokkinop@ceid.upatras.gr

scheduling algorithms requires a flexible environment where parameters, for example, the number of resources, can be easily altered and scaled. In addition, the burden of actual administrating and implementing a scheduling algorithm in an existing Grid middleware may distract the scientist from his/her main goal that is to develop efficient scheduling policies. Also, usually neither the researcher nor even the authorized personnel is permitted to alter the mechanisms of a production Grid middleware for experimental purposes, because this would distract the service provided to thousands of users. These reasons make simulation environments important tools for the evolution of Grid computing. On the other hand, the simulation environments usually hide the complexity of the Grid and the corresponding algorithms are built using assumptions that are not always realistic. Such an assumption is the *a priori* knowledge of the job workload, or the assumption that the job is divisible in smaller workloads. Also, many of these algorithms do not take into account communication or middleware delays that may hinder the scheduling algorithm's performance. As a result, simulations may not always reveal the problems that are encountered in real Grid environments. In addition, production middleware like gLite are to a large extent complex systems, developed by a closed group of researchers and, consequently, relatively little information exists on the drawbacks of the existing mechanisms, while the understanding of their behavior is limited. As a result, valuable feedback on the operation of real Grid systems is not available to the scientists who propose scheduling algorithms for these systems. Also, the requirements of Grid systems are too many, forcing the development teams to focus mainly on the most important ones, such as security, scalability, reliability, handling of heterogeneous resources, leaving the scheduling performance and related issues mostly unattended.

Our work attempts to bridge, at least to some extent, the gap between developing scheduling algorithms in simulation environments and actually building the Grid systems. For this purpose, we describe the steps, the difficulties encountered and the solutions provided to develop and evaluate a scheduling policy, initially implemented in a simulation environment, in a production Grid middleware, namely gLite (http://glite.cern.ch/). In particular, we implement in gLite the simple fair execution time estimation (SFETE) algorithm, which is a good approximation of the fair execution time estimation (FETE) algorithm [1]. SFETE attempts to allocate in a fair way the available resources among the requested jobs. An important issue we faced during the implementation was the accuracy of the information provided by the gLite's monitoring system. Accurate knowledge of the number of jobs already assigned for execution at each computational resource is key to the correct operation of SFETE and of other algorithms that base their decisions on the values of dynamically changing parameters. gLite's information service is updated at periodic intervals and, consequently, it does not always have an accurate view of the number of jobs at each resource. To address this drawback, we implemented a feedback mechanism that keeps track of the scheduler's decisions.

Another important issue we addressed is that a computation resource cannot be shared concurrently between the different users or jobs. For example, it is not possible for a job to get 30% of a resource's computational capacity and for a second one to get at the same time the remaining 70% of this capacity. Normally, jobs are executed one after the other (using the 'first come first serve' or any other ordering discipline) and each job gets 100% of the computation capacity of the resource where it is executed. This makes fairness, in the utilization of the available resources by the users, difficult to provide. For this purpose we exploit the virtualization technology and extend the gLite middleware to create and efficiently manage virtual worker nodes (WNs) that run transparently in actual nodes in the Grid infrastructure. The virtualization technology, usually implemented through software called virtual machines (VMs), makes it possible for entire 'machines' running a variety of operating systems and software to be stored on disk files, migrated over the network, and instantiated on arbitrary physical machines. VMs provide a powerful new layer of abstraction in distributed computing environments and they can increase the efficiency with which task are scheduled to the distributed resources.

We implement and evaluate the proposed mechanisms (SFETE, feedback mechanism, virtual WN mechanism) in a self-contained fully-operational testbed. Our interest is to provide a proof-of-concept implementation of these mechanisms in an existing Grid system, and show how other scheduling algorithms previously proposed in theory and evaluated through simulations, can be implemented in real Grid systems. Because the focus of our evaluation is to show experimentally the

DOI: 10.1002/cpe

feasibility and the benefits of the implemented mechanisms, we do not compare the implemented in gLite SFETE with other known from the literature algorithms but only with gLite's existing baseline scheduling policies. We believe that the implementation solutions provided can be easily applied to any other scheduling algorithm and especially to those that require up to date knowledge of dynamic information or to those that are based on reserving a percentage of the computing cycles of a resource, as SFETE does. Our experimental results show that SFETE, together with the implemented additional information update mechanism, performs better than gLite's existing scheduling algorithms, by achieving smaller job execution delay and better distribution of the jobs to the available resources. In addition, we evaluate the usage of virtual WNs along with the SFETE algorithm. Our results show that the creation of virtual WNs can enhance the performance of the Grid, by reducing the job's queuing delays. This is, to the best of our knowledge, the first time an experimental study of scheduling algorithms in gLite is performed, revealing some of gLite's weaknesses and also proposing solutions. Even though this work is specific to the gLite middleware, we believe that it may be useful to anyone interested in developing and comparing scheduling algorithms in a production Grid middleware.

The remainder of the paper is organized as follows: In Section 2 we report on previous work. In Section 3 we present a summary of the architecture of the gLite middleware. In Section 4 we describe in summary the SFETE scheduling algorithm and present the feedback mechanism implemented. In Section 5 we present the usage of virtual WNs in the gLite middleware. In Section 6 we present the experiments performed and the corresponding results. Finally, in Section 7 we conclude the paper.

2. PREVIOUS WORK

A large number of scheduling algorithms has been proposed in the Grid related literature. A taxonomy of the various scheduling algorithms is presented in [2, 3]. Job scheduling is usually performed at two levels [4]; at the higher level, a central scheduler decides the site or domain a job will be executed on, while at the lower level a local scheduler selects the exact machine where the job will be processed. In [4–6] centralized, hierarchical, or distributed scheduling schemes were presented. The proposed scheduling algorithms usually try to minimize the total average job delay [6] or maximize resource utilization, even though other performance metrics can also be used. The authors in [7] incorporated Grid economic models in scheduling algorithms that support deadline and budget constraints. Even though most Grid scheduling algorithms provide a best effort service, there are also works that try to provide hard QoS guarantees to the submitted jobs [8]. The main mechanism used for providing QoS guarantees to the Grid users is the reservation of the resources (communication, computational or storage) [9–11]. Also, fairness is not a new concept for scheduling in general, and especially for scheduling in data networks. Fairness in Grid networks has been investigated in a number works [1, 12–14]. In [12] the authors proposed a resource allocation scheme based on fair resource sharing in hierarchical virtual organizations (VOs). In [13] three different fair scheduling algorithms were proposed, called the simple fair job order policy, the adjusted fair job order policy, and the max–min fair share scheduling policy. In [14] game theory was used to prove that strong community control is required to achieve acceptable performance in Grid, by comparing centralized and distributed fair scheduling algorithms. Most of these scheduling algorithms have been evaluated through simulations, using various simulators such as Gridsim (http://www.cloudbus.org/gridsim/) [15, 16] OptorSim (http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html) [17] and others.

Scheduling is also an important issue in production Grid middleware, such as gLite [18], Globus (http://www.globus.org/) [19], UNICORE (http://www.unicore.eu/) [20], and advanced resource connector [21]. gLite is the result of the collaborative efforts of different academic and industrial research centers as part of the Enabling Grids for E-sciencE (EGEE) project [22]. EGEE brings together more than 140 institutions to produce a reliable and scalable computing resource available to the European and global research community. The EGEE infrastructure processes jobs from various scientific domains including biomedicine, high energy physics, earth sciences, astroparticle physics, computational chemistry, drug discovery, hydrology, and cosmology.

DOI: 10.1002/cpe

The gLite middleware provides high level services for scheduling and running computational jobs, for accessing and moving data, and for obtaining information on the Grid infrastructure and the Grid applications, all these functions embedded into a consistent and secure framework. The gLite related literature is relatively small in comparison to the literature regarding Grid in general and in particular scheduling in Grid networks. Most works present the functionality of the existing gLite components [23, 24], describe applications running on EGEE [25], or handle interoperability issues with other Grid middleware [26]. There are also works that attempt to model various Grid related processes, like the job arrival process and the delay introduced at different stages of job processing, using traces from gLite/EGEE and other production Grid networks [27]. These studies are important for the improved understanding of the Grid computing process.

Virtualization acquired popularity recently as a way to reduce cost of ownership and increase resources utilization. Different solutions are available to manage VMs, including Xen, VMware [28], and Kernel-based Virtual Machine (KVM). Solutions to dynamically connect virtual machines and create virtual clusters over physical resources have also appeared in the market, either in the form of services, namely Cloud computing service (e.g., Amazon Elastic Computing Cloud — EC2 [29]) or as Cloud computing toolkits (e.g., OpenNebula [30] and Nimbus [31]). The former provides resizable compute capacity as a service, while the latter allows an administrator to turn his/her cluster, data or computer center into a virtual infrastructure that provides Cloud computing services. Cloud computing is a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet. Cloud and Grids have a lot in common, but there are also differences, the most important being that Grids are typically used as batch job execution oriented infrastructures, while Clouds are more often used to provide long-time services to the end users. A comprehensive comparison between Grids (mainly focused on the EGEE infrastructure) and Clouds is presented by Conseil Européen pour la Recherche Nucléaire (French) or European Council for Nuclear Research (CERN) [32]. A number of works attempt to combine concepts and ideas from the Cloud and Grid areas, mainly through the virtualization technology, to enhance the services provided. One of the first works integrating Grid and virtualization is the Virtual Workspace [33], based on Globus, which provides a Grid execution environment that can be customized according to the users' requirements.

3. THE GLITE MIDDLEWARE

3.1. General

The gLite middleware [18] is a collection of interoperating services that cover all functionality required to provide Grid services. A typical gLite deployment involves many hosts distributed and communicating over a WAN. gLite users and resources are grouped into VOs that define both communities of users sharing a common goal and an authorization delineation of the resources accessible to each user group.

The gLite Grid services, which follow an SOA, can be grouped into five categories: access services, security services, information and monitoring services, data services, and job management services. Information and monitoring services provide a mechanism to publish and consume information on the Grid resources and their status. This information, used for scheduling, monitoring, and other purposes, is essential for the operation of the whole Grid. gLite's information system is based on the Berkeley Database Information Index (BDII). The three main services that relate to data are the storage element (SE), the file and replica catalog service, and the data management. The SE provides the corresponding of a storage resource, which can vary from simple disk servers to complex hierarchical tape storage systems. Finally, the main job management services are the computing elements (CE) and the workload management system (WMS). The CE provides the corresponding of a computation resource (e.g., cluster, supercomputers or individual workstations). A CE provides information on the underlying resources and offers a common interface for submitting and managing jobs on them. A CE has a collection of WNs, which are the nodes where the jobs actually run on. The local resource management system (LRMS) is an important component of each CE that schedules the jobs assigned to the CE on the available WNs. gLite supports the following

DOI: 10.1002/cpe

LRMSs: OpenPBS/PBSPro, Platform LSF, Maui/Torque, Batch Queueing System (BQS) and Condor. The WMS is a Grid level meta-scheduler that schedules jobs on the available CEs according to user preferences and several policies (see Section 0). It also keeps track of the jobs it manages in a consistent way, via the logging and bookkeeping (LB) service.

3.2. Workload management system

The WMS [23, 24] accepts job submission requests from users and searches for the appropriate resources where these requests should be assigned. This way the WMS hides from users the complexity of managing the Grid resources. The users simply describe the characteristics and requirements of their job requests using a high-level language, the job description language (JDL), and they submit their requests through the provided interfaces. It is then the responsibility of the WMS to match these resource requirements with the resources the users have access permissions on. The WMS also provides additional functionalities for job request management and control, such as cancellation, output retrieval, and job files perusal.

Figure 1 shows the gLite WMS architecture and the internal synergy of the WMS components. A user can access the job management services, provided by the WMS, through the workload manager proxy, using a set of tools called WMS-UI tools. The workload manager is the core component of the WMS and is responsible for selecting the most appropriate CE for the job's execution. Two main components of the WMS service are involved in the operation of WM: the matchmaker and the information super market (ISM). The matchmaker or resource broker, provides a matchmaking service, which selects the resources that best match the job's requirements. The ISM is a repository of resources information that is updated at periodic intervals and is available in read only mode to the matchmaking engine. In particular, each CE informs periodically the corresponding BDII, and the BDII informs periodically the ISM. The exact duration of these intervals can be defined by the administrator. The remainder of the WMS components handle the jobs during the rest of their lifetime. As shown in Figure 1, the WMS interacts with other basic services of the gLite middleware, such as the LB service that provides job monitoring support.

The WM, after receiving the job's requirements (that is, the JDL file), performs two operations:

- The filtering operation, through which a set of candidate CEs is created, based on the job's requirements as defined in the requirements attribute of the JDL file.
- The selection (or matchmaking) operation, where the 'best' CE is selected among the CEs in the previously created set. By 'best' we mean the CE that optimizes a metric of interest that is defined in the Rank attribute of the JDL file.

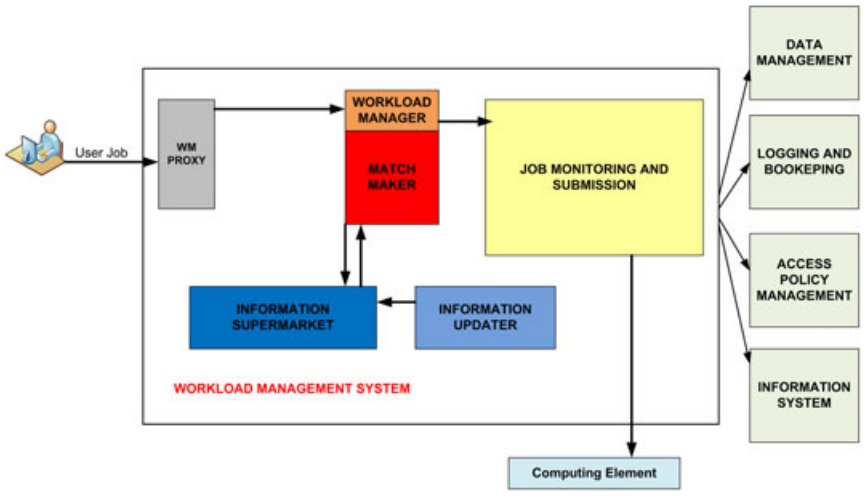

Figure 1. The gLite WMS architecture and the internal synergy of the WMS components.

DOI: 10.1002/cpe

The WMS has been designed based on the push-model philosophy where the user jobs are assigned to some specific CE early at the match-making time and then remain to the LRMS queue of the selected CE until their execution. This approach offers design scalability, minimum administration effort, and allows only 'real' jobs to be assigned to the resources (no waste of CPU cycles for other operations), which are significant advantages for a dynamic environment such as the Grid. However, the WMS operation relies on the information system for service discovery and for retrieving dynamic attributes, which leads to some disadvantages as well. Specifically, the values of the dynamic parameters are not always accurate, because there are several latencies as a result of the update rates or the information caching at the various involved levels. Furthermore, there is no support for changing the assignment of a job to some CE for execution. This is a critical issue that may lead to poor performance when, for example, there are too many jobs waiting in CE's queues for execution, given the weakness of the information system to monitor adequately dynamic parameters. This makes evident the importance of a scheduling mechanism that assigns the jobs to the available resources in a way that maximizes the resource utilization while at the same time prevents job submission to CEs that already have a lot of jobs to their queues.

Looking in the code of gLite we identified two matchmaking algorithms, which we call the Max Rank and the Fuzzy Rank algorithms. The Max Rank algorithm chooses the CE whose current state maximizes the expression at the Rank attribute. When there are more than one CEs with the same maximum Rank value, the algorithm selects randomly one of them. The Fuzzy Rank algorithm is a stochastic variation of the basic Max Rank algorithm. For each CE in the set formed by the filtering operation, Fuzzy computes the Rank expression from the JDL file. However, in this algorithm the values of the Rank expression associated to each CE represent (after normalization) the probability that each CE will be selected as the 'best' CE for the job's execution. The selection probability for a CE is therefore higher for higher ranking values. It is evident that the Fuzzy algorithm may select a different CE each time it is executed even if the user requirements and the state of the resources remain the same. Generally, the scheduling algorithm used determines to a large extent the performance of the whole Grid infrastructure, and this is why its logic and functionality are of high importance.

4. THE FEEDBACK MECHANISM

One of the most important factors affecting the performance of a Grid scheduling algorithm is the validity and the accuracy of the information provided to the algorithm for its decisions. Such information includes the computational and storage capacities of the resources, the number of the jobs assigned to the various resources, the CPU load, and others. The validity of the information becomes particularly difficult to guarantee if it changes dynamically with time, as is the case with the number of jobs running on each resource.

In Grid systems such information is provided to the scheduling algorithm by an information monitoring system. In gLite in particular, the scheduling algorithm asks the ISM service. This service, as mentioned in the previous section, is updated at periodic intervals by the BDII and, consequently, it does not always have an accurate view of the number of jobs at each resource. For information that is static, such as the computational capacity of the resources, this is not an issue, because this information does not (usually) change. The number of jobs assigned to a CE, however, is a quite dynamic parameter that changes with time, and in the time period between two ISM updates, ISM may publish outdated information for this parameter. The SFETE algorithm that we implemented in gLite, requires for its efficient operation the accurate knowledge of the number of jobs running on each resource. To handle the inaccuracies introduced by the periodic operation of the information service, we have embedded in the SFETE implementation a mechanism that provides a better estimate of the number of jobs in each CE.

4.1. Simple fair execution time estimation scheduling algorithm

We implemented from the start a new scheduling algorithm, called SFETE, in gLite, trying at the same time to identify issues that are not very visible to a theoretician designing scheduling algorithms. SFETE assigns job i to resource j that provides the minimum *simple fair execution*

DOI: 10.1002/cpe

Algorithm 1 Simple Fair Execution Time Estimation

1 **for** each job i queued in the scheduler's ordered list **do**
2 **for** each resource j in the Grid Network **do**
3 Estimate the fair execution time $\hat{X}_{ij}$:

$$\hat{X}_{ij} = w_i \cdot \frac{(N_j + 1)}{C_j}$$

End for
4 Assign job i to resource j that gives the minimum simple fair execution time $\hat{X}_{ij}$, or equivalently the minimum $(N_j + 1)/C_j$
5 Send the scheduling decision to the user of job i.
end for

time $\hat{X}_{ij}$. The simple fair execution time $\hat{X}_{ij}$ is an estimation of the time by which job i will be executed on resource j, assuming it gets a fair share of the resource's computational power, without taking into account the fair execution times of the other jobs assigned to the resource.

The simple fair execution time $\hat{X}_{ij}$ of job i on resource j is defined as

$$\hat{X}_{ij} = w_i \cdot \frac{(N_j + 1)}{C_j} \tag{1}$$

where w_i is the workload of job i, C_j is the computational capacity of resource j, and N_j is the number of jobs in the resource's queue, including the one being processed at the time. It is important to note that the calculation of the simple fair execution time $\hat{X}_{ij}$ of job i on resource j is only an estimate. New jobs may be sent to resource j, or existing jobs may complete their execution, changing the number of pending jobs and, consequently, the fair share of the computation capacity the jobs already assigned to the resource should take; however, in SFETE the simple fair execution time estimations of the jobs are not reestimated when such changes occur. Also, we should note that the *a priori* knowledge of workload w_i is not needed for SFETE, because the optimization in (1) is made over j (the resource where job i should be sent to). The pseudocode of the SFETE algorithm is presented in Algorithm 1.

4.2. Implementation

In this section we describe the steps we followed to implement SFETE in the gLite Grid middleware. In order for a developer to be able to extend and build the components of gLite, the ETICS system is required. ETICS, which stands for eInfrastructure for Testing, Integration and Configuration of Software, is an online collaborative service for managing software projects by managing their configuration, enforcing quality standards, building packages and testing them in environments as close as possible to real-world infrastructures. Through ETICS the developer can download the modules, files, libraries, and configuration files needed to build the component of interest (e.g., the WMS). Once the code and a suitable configuration have been downloaded, the developer can use the ETICS client to generate software packages and reports. For our development purposes we downloaded the WMS code using the *etics-checkout* command with the appropriate parameters:

```
etics-checkout --platform slc4_ia32_gcc346 -c glite-wms_R_3_2_1_20 org.glite.wms
```


DOI: 10.1002/cpe

After downloading the code we were able to compile and build it locally, in our development machine, using the command *etics-build*:

etics-build -c glite-wms_R_3_2_1_20 org.glite.wms

This command produces several files containing information about the results of the compilation. Also, the compilation creates RPM Package Manager installation files for every functional component in a separate folder. These RPM files can be used for updating an existing installation of the gLite middleware with the implemented changes. This way the user is alleviated of the burden of caring about the various code interdependencies that may exist. We also followed this approach by updating the WMS/LB node of our testbed with the RPMs we created.

Simple fair execution time estimation uses the two-phase procedure (filtering and selection) described previously (Section 3.2), to select the appropriate CE for a given job. For the filtering phase, SFETE uses the same strategies as the Max and the Fuzzy Rank algorithms. For the selection phase, SFETE computes the simple fair execution time estimation for a given job at each filtered CE, and assigns the job to the CE that gives the minimum value. Therefore, in SFETE the ranking is not based on the user preferences as they are expressed by the Rank attribute of the JDL file, but on the minimum simple fair execution time estimation. If there are more than one computation resources that yield the same simple fair execution time estimation, the job is assigned randomly to one of them. For a user to be able to use the SFETE algorithm, we defined a new boolean JDL attribute, called *FeteRank*.

An important part of the SFETE implementation is the calculation of the simple fair execution time. For this calculation SFETE must know for each filtered CE, its computation capacity and the number of jobs assigned to it. To obtain the computation capacity information, SFETE asks the ISM service. Each resource in the Grid publishes various information, including its total number of CPUs and their clock speed, and also information on the speed rating of each CPU based on the nominal SpecInt2000 (http://www.spec.org/cpu2000/) benchmark. Each of these CPU characteristics can be used for expressing the computational capacity of the resource. On the other hand, the number of jobs assigned to a CE is a dynamic property, which changes with time. As a result, in the time period between two updates the ISM may publish outdated information for the number of jobs in a CE (and for other dynamic information). However, in order for the SFETE algorithm to operate efficiently, it must know with some accuracy this information. For this reason we have embedded in the SFETE implementation a mechanism that provides a better estimate of the number of jobs in a CE, addressing in this way the intrinsic weakness of the information system to provide accurate information for dynamic parameters. The basic idea of the mechanism is the use of a local counter for each CE, which is updated from the workload manager every time a job is assigned for execution to the corresponding CE. This value is of course only an estimate, because the local counters are informed only for the jobs that are assigned to each computation resource and not for the jobs that complete their execution in the time period between two ISM updates. For this reason in every ISM update, the local counters are updated with the data published from the ISM service.

In summary, for the implementation of SFETE in gLite we performed the following changes in the code:

- **org.glite.wms.broker:** In the *src/selectors/RB-SelectionSchema.cpp* file we defined the existence of the SFETE selection strategy. In the same directory we created the SFETE related files (*SFeteRankSelector.h* and *SFeteRankSelector.cpp*).
- **org.glite.wms.matchmaking:** In the file *matchmakerISMImpl.cpp* we implemented the *SFeteRank()* function.
- **org.glite.wms.helper:** In the *src/broker/Helper_ ism.cpp* file and in the *flatten_requirements()* function, we call the *SFeteRankSelector* function in the case where the *SFeteRank* attribute is set in the JDL file.
- **org.glite.wms.ism:** In the *ism.cpp* and in the *purchaser/ldap-utils.cpp* files we implemented the internal feedback mechanism that keeps track of the jobs that have been assigned to the available CEs.

DOI: 10.1002/cpe

5. VIRTUAL WORKER NODES MECHANISM

In most current Grid middleware, including gLite, a resource cannot be shared concurrently between different users or jobs, mainly because of the capabilities of the underlying operating systems. This means that a job uses 100% of a resource until its execution is completed, while it is not possible for two jobs to use concurrently, for example 60% and 40%, respectively, of a resource's computational capacity for the whole duration of their execution. As a result, advanced scheduling policies proposed and evaluated in simulation environments cannot find direct application in the existing Grid middleware. In our work we examine how this drawback can be handled in gLite by using the virtualization technology to increase the Grid network's resource availability and performance.

In particular, we extended the gLite middleware to create and efficiently manage virtual WNs that run on actual nodes of the Grid infrastructure and seem and behave as actual WNs to the other gLite services. The virtual WNs are in fact VMs that have specific characteristics (operating system, CPU capacity, memory size, disk size). The virtual WNs are transparent to all the gLite services, to the end users, to the administrators, and to the applications, except for the WMS and the local CE scheduler that are responsible for their dynamic creation, destruction, and reallocation of available computational capacity, as new jobs arrive or jobs complete their execution. The same approach can also be used to provide services such as resource fault tolerance, job migration, and others, which however were not considered or implemented in this work. Moreover, as a result of recent progress in VM technology, these described functionalities no longer come at a performance cost to either the application or the hosting resource; systems such as Xen demonstrate that they can be used with little or no performance degradation.

Although we build the proposed virtual WN mechanism in a generic way as to be used by any scheduling policy, we focus on how it can enhance the functionality of the SFETE fair scheduling algorithm presented earlier (Section 4.1). Our purpose is to take advantage of existing technologies/mechanisms implemented in gLite, incorporating our virtual WN mechanism in a way that is as transparent as possible to the other services.

5.1. Simple fair execution time estimation enhancements

Simple fair execution time estimation is extended through the virtual WN functionality as follows. For each new job i requesting service the scheduler calculates its fair execution time X_{ij} if executed on resource j, for all j, and assigns it to the resource that gives the minimum fair execution time. Next, the scheduler orders the components of the virtual WN mechanism, residing in the chosen resource, to start a new virtual WN in this resource with computational capacity equal to

$$\frac{C_j}{N_j(t)+1'}$$

where C_j, is the computational capacity of the resource and $N_j(t)$ is the total number of jobs already assigned to it, at the time t the assignment decision is made. In addition, our virtual WN mechanism readjusts (decreases) the computational capacities of all the virtual WNs already running in the resource accordingly. This means that in practice all the virtual WNs in a resource get an equal share of the resource's computational capacity. This readjusting procedure also occurs when a job completes its execution, in which case its virtual WN is closed and the computational capacities of all the other virtual WNs are again readjusted (increased). Therefore, through the virtualization technology it becomes possible to run concurrently more than one job on the same physical machine and also to adjust on demand the percentage of CPU each virtual WM takes, making time sharing possible.

Figure 2 shows the way virtual WNs are created and closed and the way their computational capacity is allocated and reallocated as new jobs arrive at the resource or existing jobs complete their execution. Task 1 is the first task arriving at the resource and the corresponding virtual WN created has the maximum possible computation capacity, equal (theoretically at least) to that of the underlying physical machine. At time instance t_1, a new task arrives (Task 2) and a new virtual WN is created for serving it. In this case, the computation capacity of the first virtual WN is reduced to 50% of the resource's maximum capacity, while the second virtual WN receives the remaining

DOI: 10.1002/cpe

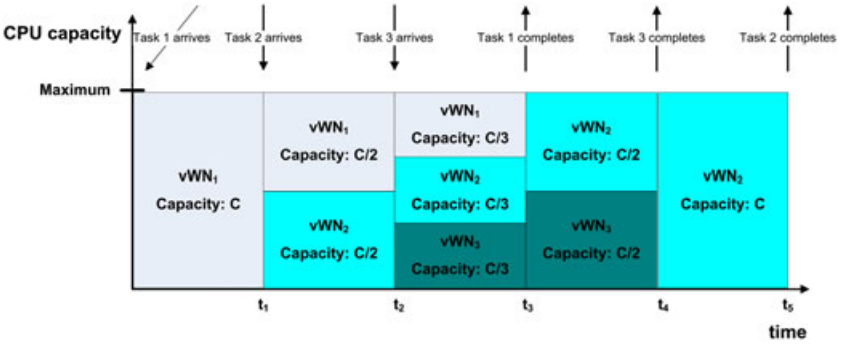

Figure 2. The virtual WNs and their computational capacities as new jobs arrive in the resource or complete their execution.

50%. At time instance t_2, a third task arrives (Task 3), resulting in the creation of another virtual WN and in the adjustment of the computational capacities of all the existing virtual WNs to be equal to 33% of the resource's maximum capacity. At time instance t_3, Task 1 completes its execution, its virtual WN is closed and its computational capacity is reallocated equally among the remaining virtual WNs. At time instance t_4, Task 3 also completes its execution, leaving in the system only one virtual WN with maximum computational capacity. In the end, Task 2 also completes and all the virtual WNs are closed.

We should stress that the reestimation of the fair execution times when new tasks arrive or existing tasks complete their execution at a resource is easily implemented in a simulation environment, but not in a production Grid middleware, like gLite. This is also the case for any scheduling algorithm using a dynamically varying ranking metric for selecting the resource where a task should be assigned. This is because the reestimation of a metric (e.g., fair execution time) may change the relative priorities of the tasks, and may lead to the need of stopping some tasks in favor of others, or even moving some tasks to other resources. However, this functionality is not available today, at least in gLite, using the existing WMS. The virtual WNs mechanism we propose handles the reestimation of the tasks' fair execution times required when task arrivals or departures occur, by readjusting the virtual WNs' computational capacities. However, more extensions (not considered in this work) are required, such as adding the capability to the WMS to pause the execution of tasks, to move them to other machines, etc. These extensions are made possible by the virtual WNs mechanism presented in this work, because it is easier to move a virtual machine between physical machines, instead of moving a task with its intermediate states.

5.2. Implementation

The virtual WN mechanism consists of several dedicated middleware components/services that are spread over all the levels of the gLite hierarchy (Figure 3). Through this mechanism, the virtual WNs at the various resources/sites are initiated and managed dynamically. The users simply describe the characteristics and requirements of their job requests as usually, using JDL, and submit their requests through the provided gLite interfaces. Then it is the responsibility of the WMS and of the virtual WN mechanism to determine which resource each job will be assigned to, which virtual WN will be initiated and with what computational capacity. Also, the virtual WN mechanism is responsible for terminating virtual WNs when the corresponding jobs complete their execution and for reallocating the available capacity to the remaining virtual WNs at a resource.

More particularly, the implemented mechanism consists of the following components/services (Figure 3):

- WMS level services: interact with some of the core services of the WMS. Through this interaction our mechanism is able to manage the creation of the virtual WNs at the CE.
- CE level services: their primary purpose is to manage the life-cycle of the virtual WNs that belong to a particular CE. These services receive operational commands from the WMS level

DOI: 10.1002/cpe

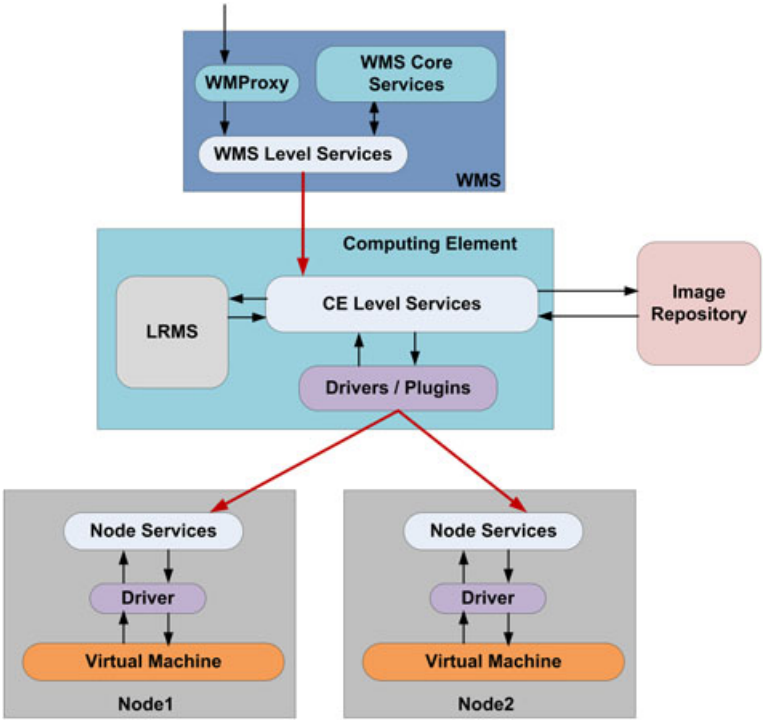

Figure 3. Virtual WN mechanism's components and their interaction.

and act accordingly. Furthermore, they orchestrate the operation and interaction between the other components of gLite or of our mechanism, such as the LRMS, and the Image Repository.

- Node services: these services receive operational commands by the CE level and act accordingly upon the running virtual WNs.
- Image Repository: holds preconfigured images that are used as virtual WNs.

6. PERFORMANCE RESULTS

6.1. Simulation results

Initially, we compare SFETE scheduling algorithm against some well-known algorithms (Table I), using the GridSim [15] simulator. The scheduling algorithms were implemented in a centralized and 'offline' manner, where all (waiting to be scheduled) tasks are first ordered and then assigned to the appropriate resource. In the earliest deadline first (EDF) ordering policy the task with the most imminent deadline is scheduled first, while in the least length first (LLF) ordering policy, the task with the smallest workload is given priority. The earliest completion time (ECT) assignment policy, assigns a task to the resource where the task will finish its execution earlier. Also, the SFETE algorithm uses the first come first serve (FCFS) ordering policy, where tasks are processed (assigned to resources) in the order they arrive at the scheduler.

Table I. The scheduling algorithms compared with the SFETE.

Algorithm	Ordering policy	Assignment policy
FCFS/ECT	FCFS	Earliest completion time (ECT)
EDF/ECT	EDF	Earliest completion time (ECT)
LLF/ECT	LLF	Earliest completion time (ECT)

DOI: 10.1002/cpe

All the scheduling algorithms were evaluated in a nonuniform resource scenario, in which the resources have different characteristics. The total computational capacity of the resources in both scenarios was the same. The tasks characteristics are defined probabilistically and the users' task submission rate follows an exponential distribution, whose mean takes the following values: 12, 20, 25, 33, 40, 50, 55, 60, 65, 70 tasks/s. Also, in our simulations we assume that the communication delays are negligible compared with the execution time of the tasks, which is the case in computational Grids.

We used the following metrics:

- **Average task delay:** The average of the delays of the tasks (task Delay = task Finish time – task Creation time).
- **Average excess time:** The average time by which a task misses its noncritical deadline (task Excess time = task Finish time – task Deadline expiration).
- **Deadlines missed:** The number of tasks that miss their noncritical deadlines.

For all the scheduling algorithms examined the average task delay increases as a function of the task submission rate (Figure 4). Specifically, for light load all the algorithms have similar behavior; however, when the task submission rate increases the SFETE algorithm achieves smaller average task delay. This happens because the proposed algorithm treats the tasks and utilizes the resources in a more fair manner, something that becomes more evident as the task load increases. We also observe that the SFETE algorithm results in smaller task delay standard deviation than the other algorithms (Table I).

Figure 5 illustrates that the average excess time increases as a function of the task submission rate; this increase is smaller when the SFETE algorithm is used, meaning that the times by which the tasks miss their deadlines are also smaller.

Finally, our performance results showed (Figure 6) that fewer tasks miss their deadlines when they are scheduled using the SFETE algorithm than when they are scheduled with other algorithms.

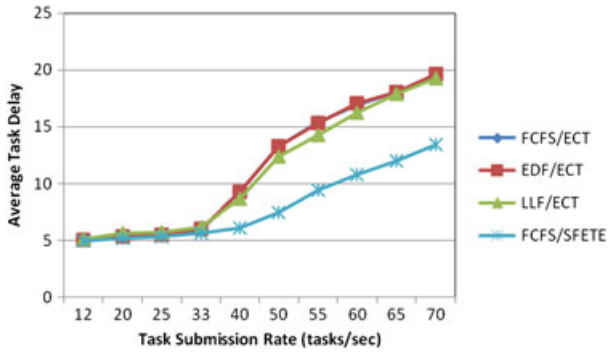

Figure 4. Average task delay versus task submission rate.

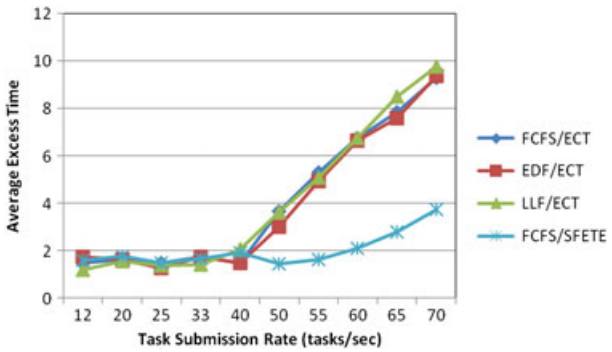

Figure 5. Average excess time versus task submission rate.

DOI: 10.1002/cpe

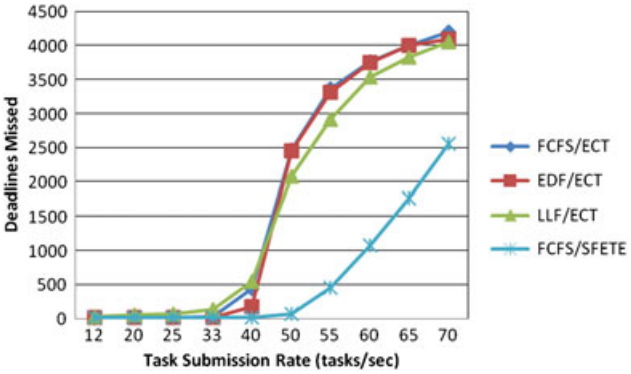

Figure 6. Deadlines missed versus the task submission rate.

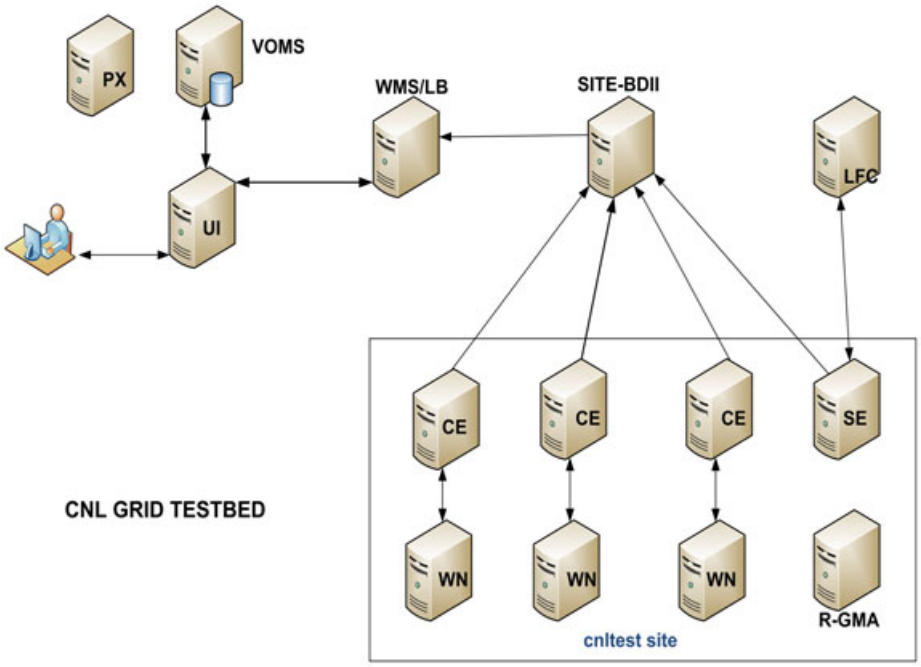

Figure 7. The gLite testbed used in our experiments.

This is due to the fact that resources are utilized more uniformly, something that becomes more evident as the task load increases.

6.2. Testbed results

To evaluate the performance of the SFETE scheduling algorithm when it is implemented in a production grid middleware, we set up a small Grid infrastructure based on the gLite middleware. This way we were able to evaluate SFETE in a realistic instead of a simulation environment and compare it against the scheduling algorithms that gLite middleware provides by default to its users. Figure 7 shows the Grid infrastructure we set up based on gLite 3.1 middleware.

It is beyond the scope of this work to present in detail the installation and configuration process of the testbed nodes; in what follows, we highlight instead the basic steps we followed. Initially, we decided the exact topology of our Grid infrastructure, because this information is necessary during the configuration of the various gLite nodes. Next, we installed the Scientific Linux (http://www.scientificlinux.org/) operating system to each node and time synchronized them using

DOI: 10.1002/cpe

the Network Time Protocol (NTP) protocol. Moreover, for all gLite nodes, except the UI, WN, and BDII nodes, we installed the necessary host certificate/key files. These files must be issued by a trusted (for the gLite middleware) certification authority (CA). Next, we installed the gLite middleware software packages, by selecting the appropriate software repositories and downloading the corresponding software. After the successful software installation we configured each node using the yaim tool. To configure each node, we edited, when needed, several configuration files and then executed the yaim tool. As mentioned, in order for the authentication to work, all users and services need to have a certificate issued from a trusted CA. In our testbed we did not use a commercial CA, but established our own trusted CA. In particular, we implemented our CA using the OpenSSL command-line tool that provides all the functionalities required. This approach is recommended only for small infrastructures that are not connected to the Internet.

Our testbed has only one site that has four CEs (Table II) for executing the jobs. The CEs use as LRMS the Torque/PBS system. The BDII informs the ISM component of the WMS on the status of the various CEs. The WMS we used was modified because we have added to it the SFETE scheduling algorithm. Virtual Organization Membership Service (VOMS) provides information such as VO membership, group membership, and the roles of each user. In our experiments we used a single VO that gets 100% of each CE's computational capacity. Finally, through the UI each user can be authenticated and authorized to use the Grid resources.

6.2.1. Experiment parameters. We performed a number of experiments to evaluate the performance of the SFETE algorithm and the proposed mechanisms against that of the default gLite scheduling algorithms. In our experiments we have five users, with the same rights, who submit jobs to the Grid testbed. The user's job submission process is assumed to be Poisson with the following average arrival rates: 5, 10, 15, 20, 25, and 30 jobs/min. On the basis of the work we performed in [27] we observed that in the EGEE infrastructure job interarrival times follow an exponential distribution and therefore Poisson is a good approximation for the job arrival process. We do not use larger arrival rates because our Grid testbed has limited computation resources and larger rates would overload it. Users generate jobs of different mean durations: 5, 6.5, 8.5, 10.5, and 12.5 min. To be more precise, this is the duration of the corresponding job when it is executed at the highest computational capacity resource, without competing with any other jobs for the resource. The jobs created did not have any data dependencies, because this would complicate our study.

In our experiments we used a default filtering strategy, by setting the Requirement attribute of the JDL file to 'Production', where all the CEs are selected. For the selection strategy the Max and Fuzzy Rank algorithms select a CE based on a dynamic expression at the Rank attribute of the JDL file, where a job is submitted to the computation resource that has the fewest jobs in its queues. On the other hand SFETE selects the CE with the minimum fair execution time estimation. This way the dynamic ranking used affects only the decisions of the Max and Fuzzy Rank algorithms, and not those of SFETE. In our experiments we used the following metrics to compare the performance of the Max, Fuzzy Rank, and SFETE scheduling policies:

- The average job total delay at the Grid, measured from the time a job is created until the time it completes its execution at a CE.
- The makespan, the time interval between the submission of the first job at the Grid infrastructure and the time when the last job finishes its execution at some CE.

Table II. The characteristics of the storage and computational resources of our testbed.

Resource type	Details
SE	Pentium 4 1.6 GHz Hard disk 60 GB
CE/WN 1	Ram 512 MB Pentium 4 1.8 GHz
CE/WN 2	RAM 512 MB Pentium 4 3.0 GHz
CE/WN 3	RAM 512 MB Intel Core 2 DUO 3.0 GHz
CE/WN 4	RAM 1 GB Intel Core 2 DUO 3.0 GHz

DOI: 10.1002/cpe

- The average time a job remains queued at a CE before its execution.
- The distribution of jobs at the available CEs.

6.2.2. Results. Initially, we performed experiments evaluating SFETE along with the feedback mechanism without the virtual WNs mechanism. Figure 8 shows the average job total delay as a function of the job submission rate. For all the scheduling algorithms examined, the average job total delay increases as the job submission rate increases because queuing delays increase. SFETE achieves smaller average total delay than the Max and Fuzzy Rank algorithms for all the submission rates examined, as indicated by the results presented in Figure 8(a). Figure 8(b) shows the makespan for all the algorithms. This metric depends on the maximum job delay at the queues of the CEs and the average job execution time. As we can see SFETE outperforms the other gLite algorithms.

Figure 9 shows that the SFETE scheduling algorithm results in a rather uniform distribution of the jobs to the available computational resources, for all job submission rates. SFETE assigns a job to the computational resource that provides, at the time of the submission, the minimum simple

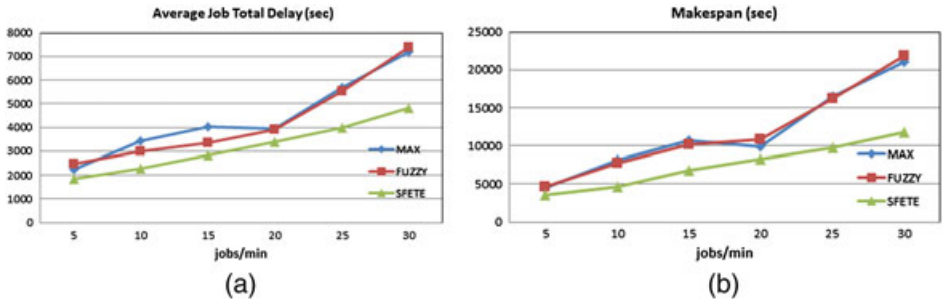

Figure 8. (a) The average job total delay and (b) the makespan for the Max Rank, the Fuzzy Rank, and the SFETE algorithms as a function of the job submission rate.

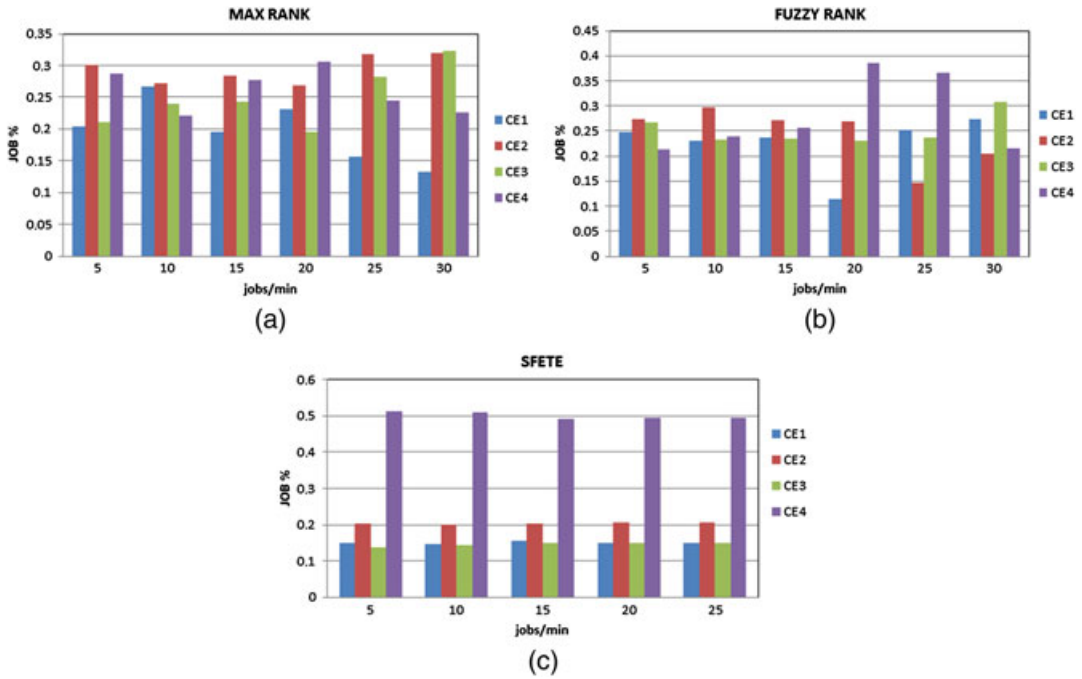

Figure 9. The job percentage distributions at the available CEs that (a) Max Rank, (b) Fuzzy Rank and (c) SFETE achieve as a function of the job submission rate.

DOI: 10.1002/cpe

fair execution time estimation. The result of this process is independent of the job submission rate, because SFETE assigns to each available resource a percentage of the total number of jobs that is proportional to its computational capacity. In the Max and Fuzzy Rank algorithms the distribution of the jobs changes randomly with the job submission rate and there is always a chance that a resource will be overloaded, becoming a bottleneck for the whole system.

These results are mainly due to the fact that both the Max and the Fuzzy Rank algorithms use the information provided by the ISM to evaluate the Rank expression at the job's JDL file. The Rank expression for these algorithms favors resources (CEs) that have a small number of jobs in their queues. However, as already mentioned, the information that ISM publishes is updated at periodic intervals. In the time period between two updates the ISM and, consequently, the scheduling algorithms are not aware of new jobs arriving at the resources for execution. For example, it is possible that in this time period a large number of jobs may be assigned to a previously light-loaded resource, overloading it, without the ISM acknowledging that. As a result, the decisions that the Max and Fuzzy Rank take are based on outdated information and may be incorrect. The effects of outdated information become even more pronounced as the job submission rate increases. On the other hand, the mechanism we implemented in SFETE provides a better estimate of the number of jobs at the resource queues. As a result, the SFETE scheduling algorithm always maintains a good distribution of jobs at the available computation resources, independently of the job submission rate.

Figure 10 shows the average time a job remains at the LRMS queues of the CEs as a function of the job submission rate. This time is proportional to the total number of jobs that use the same CE and affects the average execution time of a job. Again, SFETE outperforms the Max and the Fuzzy Rank algorithms, because it better distributes the jobs over the available resources.

We also performed experiments evaluating the SFETE algorithm enhanced with the virtual WN mechanism. During our evaluation we observed that if the number of virtual WNs created in each resource were unlimited, then a huge number of virtual WNs were created at each resource. These virtual WNs and in particular the corresponding VMs require a small portion of the resource's computational capacity for their operation, even when not executing any job. As a result, when too many virtual WNs are created at a resource, the actual free computational capacity of the resource is reduced to the extent that it becomes incapable of serving new jobs. For this reason, we performed experiments by altering the maximum number of allowed virtual WNs per resource. When more jobs are assigned to a resource than the resource's maximum number of allowed virtual WNs, then some of these jobs are assigned to the already created virtual WNs. From the experiments conducted we observed that the average job queuing delay is reduced (Figure 11) as a function of the maximum number of allowed virtual WNs. On the other hand the average job delay is not reduced and it even increases slightly. This is because independently of whether the virtual WN mechanism is used or not, the same number of jobs try to use the same amount of resources, while the used VMs also use a small portion of the resource's computational capacity. Therefore, what our proposed virtual WN mechanism actually achieves is the flexible (fair in our case) use of the available resources (for example, jobs can get execution cycles from the time they arrive at the resource). This flexibility can become more visible when different computational capacities are assigned to the virtual WNs,

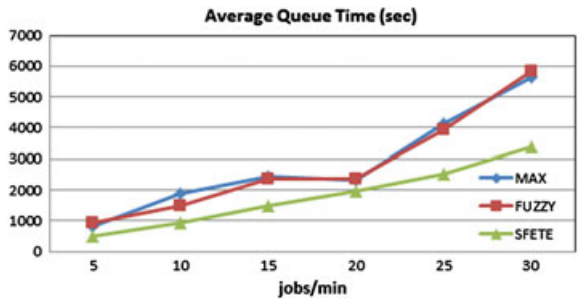

Figure 10. The average time jobs remain at the queues of the CEs for the Max Rank, the Fuzzy Rank and the SFETE algorithms as a function of the job submission rate.

DOI: 10.1002/cpe

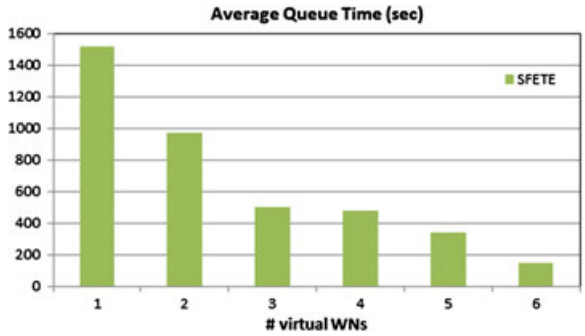

Figure 11. The average time jobs remain at the queues of the CEs using the SFETE algorithm enhanced with the virtual WN mechanism as a function of the maximum number of allowed virtual WNs per resource and for job submission rate equal to 15 jobs/s.

or when a job's execution in a virtual WN is allowed to pause to execute in another virtual WN a job with a higher priority. In the present work, we did not study these cases, because they were not in the same context as SFETE.

7. CONCLUSIONS

Incorporating a new scheduling algorithm in a production Grid middleware, such as gLite, may not be as straightforward as in the case of developing it in a simulation environment, for example GridSim. This is because assumptions and simplifications usually made in simulations do not hold in reality. We identified two important such implementation issues, namely the inaccuracy of the information provided to the scheduler by the information system, and the inflexibility in the sharing of a resource among different jobs, and we developed corresponding mechanisms to address them. Our results indicate that simple changes in the gLite's scheduling procedures can yield significant performance gains. We believe that the mechanisms implemented, together with other changes, will make it easier to incorporate in gLite other scheduling algorithms previously proposed only in theory. We also verified that the use of virtualization technology is very promising, because resources can be shared in a more flexible way among the jobs. However, it should be used with care because it cannot, of course, reduce the execution time of the jobs, while there is an overhead associated with the creation of virtual WNs. Finally, we believe that this work is a good starting point for anyone interested in extending, improving, and evaluating the gLite's scheduling mechanisms.

ACKNOWLEDGEMENT

This work was supported by the European Middleware Initiative (EMI) (http://www.eu-emi.eu/).

REFERENCES

1. Dafouli H, Kokkinos P, Varvarigos E. Fair execution time estimation scheduling in computational grids. *International Conference on Distributed and Parallel Systems (DAPSYS)* 2008:93–104.
2. Krauter K, Buyya R, Maheswaran M. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience* 2002; **32**(2):135–164.
3. Kertesz A, Kacsuk P. A taxonomy of grid resource brokers. *International conference on distributed and parallel systems (DAPSYS)* 2007:201–210.
4. Braun T, Siegel HJ, Beck N, Bölöni LL, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B, Hensgen D, Freund RF. A comparison of eleven static heuristics for mapping a class of independent jobs onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing* (June 2001); **61**(6):810–837.
5. Dail H, Casanova H, Berman F. A decoupled scheduling approach for grid application development environments. *Journal of Parallel and Distributed Computing* (May 2003); **63**(5):505–524.
6. Cardinale Y, Casanova H. *An evaluation of Job Scheduling Strategies for Divisible Loads on Grid Platforms*. High Performance Computing and Simulation Conference (HPCS), 2006.

DOI: 10.1002/cpe

7. Buyya R, Abramson D, Giddy J, Stockinger H. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice & Experience (CCPE)* 2002; **14**(13-15):1507–1542.
8. Kokkinos P, Varvarigos E. A framework for providing hard delay guarantees and user fairness in Grid computing. *Journal of Future Generation Computer Systems* 2009; **25**(6):674–686.
9. Smith W, Foster I, Taylor V. Scheduling with advanced reservations. *IPDPS* 2000:127–132.
10. Castillo C, Rouskas GN, Harfoush K. On the design of online scheduling algorithms for advance reservations and QoS in grids. *IPDPS* 2007:1–10.
11. Stevens T, De Leenheer M, Develder C, Dhoedt B, Christodoulopoulos K, Kokkinos P, Varvarigos E. Multi-cost job routing and scheduling in Grid networks. *Journal of Future Generation Computer Systems* 2009; **25**:912–925.
12. Kim KH, Buyya R. Fair resource sharing in hierarchical virtual organizations for global grids. *In Proceedings of the 8th IEEE/ACM International Conference on Grid Computing (GRID '07). IEEE Computer Society*, Washington, DC, USA, 2007; 50–57.
13. Doulamis N, Varvarigos E, Varvarigou T. Fair scheduling algorithms in grids. *IEEE Transactions on Parallel and Distributed Systems* 2007; **18**(11):1630–1648.
14. Rzadca K, *et al.* Fair game-theoretic resource management in dedicated grids. *IEEE/ACM CCGrid* 2007:343–350.
15. Buyya R, Murshed M. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience (CCPE)* 2002; **14**(13-15):1175–1220.
16. Sulistio A, Cibej U, Venugopal S, Robic B, Buyya R. A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurrency and Computation: Practice and Experience (CCPE)* 2008; **20**(13):1591–1609.
17. Cameron D, Millar A, Nicholson C, Carvajal-Schiaffino R, Zini F, Stockinger K. Optorsim: a simulation tool for scheduling and replica optimisation in data grids. *Computing in High Energy and Nuclear Physics* 2004.
18. Laure E, Fisher SM, Frohner Á, Grandi C, Kunszt P, Krenek A, Mulmo O, Pacini F, Prelz F, White J, Barroso M, Buncic P, Hemmer F, Di Meglio A, Edlund A. Programming the Grid with gLite. *Computational Methods in Science and Technology* 2006; **12**:33–45.
19. Foster I. Globus Toolkit version 4: Software for service-oriented systems. *IFIP International Conference on Network and Parallel Computing*, Beijing, China, 2005; 2–13.
20. Erwin D. UNICORE - A Grid computing environment. *Concurrency and Computation: Practice and Experience (CCPE)* 2003; **14**:1395–1410.
21. Ellert M, Grønager M, Konstantinov A, Kónya B, Lindemann J, Livenson I, Nielsen J, Niinimäki M, Smirnova O, Wäänänen A. Advanced resource connector middleware for lightweight computational Grids. *Journal of Future Generation Computer Systems* 2007; **23**(2):219–240.
22. Enabling Grids for E-sciencE (EGEE) project web site. http://www.eu-egee.org/.
23. Andreetto P, *et al.* The gLite workload management system. *Journal of Physics: Conference Series* 2008; **119**(6).
24. Cecchi M, Capannini F, Dorigo A, Ghiselli A, Giacomini F, Maraschini A, Marzolla M, Monforte S, Pacini F, Petronzio L, Prelz F. The gLite workload management system. *International Conference on Grid and Pervasive Computing*, Geneva, Switzerland, Lecture Note in Computer Science, 2009; 256–268.
25. Montagnat J, *et al.* A secure grid medical data manager interfaced to the gLite middleware. *Journal of Grid Computing (JGC)* 2008; **6**(1):45–59.
26. Grønager M, *et al.* Interoperability between ARC and gLite - Understanding the Grid-Job Life Cycle. *IEEE international Conference on Escience*, Indianapolis, IN, 2008; 493–500.
27. Christodoulopoulos K, Gkamas V, Varvarigos EA. Statistical analysis and modeling of jobs in a grid environment. *Journal of Grid Computing* 2008; **6**(1):77–101.
28. VMware: http://www.vmware.com/.
29. Amazon Elastic Cloud – EC2: http://aws.amazon.com/ec2/.
30. OpenNebula: http://www.opennebula.org.
31. Nimbus: http://www.nimbusproject.org/.
32. An EGEE Comparative Study: Grid and Clouds - Evolutions or Revolution? (CERN) https://edms.cern.ch/document/925013/.
33. Keahey K, Foster I, Freeman T, Zhang X, Galron D. *Virtual Workspaces in the Grid*. Euro-par: 421–431, 2005.

DOI: 10.1002/cpe