Developing scheduling policies in gLite middleware

A. Kretsis, P. Kokkinos, E. A. Varvarigos
Computer Engineering and Informatics Department, University of Patras, Greece
Research Academic Computer and Technology Institute, Patras, Greece
{akretsis, kokkinop, manos@ceid.upatras.gr}

Abstract

We describe our experiences from implementing and integrating a new job scheduling algorithm in the gLite Grid middleware and present experimental results that compare it to the existing gLite scheduling algorithms. It is the first time that gLite scheduling algorithms are put under test and compared with a new algorithm under the same conditions. We describe the problems that were encountered and solved, going from theory and simulations to practice and the actual implementation of our scheduling algorithm. In this work we also describe the steps one needs to follow in order to develop and test a new scheduling algorithm in gLite. We present the methodology followed and the testbed that was set up for the comparisons. Our research sheds light on some of the problems of the existing gLite scheduling algorithms and makes clear the need for the development of new.

1 Introduction

The emergence of high speed networks is making the vision of Grids a reality. 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 information services, resource discovery and monitoring, job submission and management, brokering, data management and resource management.

A number of production Grid middlewares exist today, such as gLite [11], the Globus Toolkit 4 (GT4) [13] and UNICORE [14]. The 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 [12]. The EGEE infrastructure processes jobs from various scientific domains, including High Energy Physics (HEP), Biomedicine, Earth Sciences and others.

An important part of gLite and of any Grid middleware is job scheduling. Scheduling is a key to the success of Grid Networks, since it determines the efficiency in the use of the resources and the Quality of Service (QoS) provided to the users. The scheduling of jobs to resources has been considered, among others works, in [1],[2],[3], where several centralized, hierarchical or distributed scheduling schemes are presented. Other works incorporate economic models in Grid scheduling, as in [4], which proposes scheduling algorithms that take into account deadline and budget constraints. Fair scheduling in Grid networks has also been addressed in [5],[6]. Most of these scheduling algorithms, have been evaluated through simulations. A taxonomy of the existing Grid resource schedulers is presented in [7],[8].

The gLite's development is performed by a close group of researchers and as a result little information exists on how to extend or change its functionality. Furthermore, since gLite is a production Grid middleware, it consists of a large number of components, whose installation and configuration can be quite difficult. We believe that one of the main contributions of this work is that it gives some insight on how gLite code is structured and especially the code of gLite's scheduling component. So, our work can be a starting point for anyone interested in extending the gLite middleware. Furthermore, it is, to the best of our knowledge, the first time that gLite scheduling algorithms are put under test and compared with a new algorithm under the same conditions, revealing some of their weaknesses.

In our work we implement in gLite the Simple Fair Execution Time Estimation (SFETE) [9] algorithm, which is a good approximation of Fair Execution Time Estimation (FETE) algorithm. FETE assigns a job to the resource that minimizes what we call its fair execution time estimation. The fair execution time of a job at a resource is obtained assuming that the job gets a fair share of the resource's computational power. Though space-shared scheduling is used in the actual system, the estimates of the fair execution times are found assuming time-sharing (processor sharing) is used. The main difference between FETE and SFETE is that the second algorithm does not require a-priori knowledge (or estimates) of the job workloads, which would be an important requirement when implementing a scheduling algorithm in a real Grid middleware. In [9] we performed an extensive set of simulation experiments, comparing FETE and SFETE. The results showed that the FETE and the SFETE algorithms give similar results, while both outperforming a number of known scheduling algorithms. In this work we implemented SFETE algorithm in the gLite middleware and evaluated its performance in a self-contained testbed against the existing gLite scheduling

978-0-7695-3622-4/09 $25.00 © 2009 IEEE
DOI 10.1109/CCGRID.2009.54

algorithms. gLite implements two scheduling algorithms, which we call Max Rank and Fuzzy Rank. Our results show that SFETE performs better than the other two scheduling algorithms, by achieving smaller job execution delay and better distribution of the jobs to the available computational resources. A key point for the correct operation of SFETE is the accurate knowledge of the number of jobs that are already assigned for execution at each computation resource. gLite's information service is updated at periodic intervals and as a result it does not always have an accurate view of the number of jobs at each resource. We will show that this is an important drawback, hindering the performance of the existing algorithms. To address this drawback, we implemented a mechanism in SFETE that provides a good estimate of this value. Though this work is specific to gLite middleware, we believe that it may be useful for anyone developing and comparing scheduling algorithms in a production Grid middleware. Issues like testbed set up, experimental methodology, metrics measured and the need for an accurate information service are common in any kind of Grid middleware. The description we give on the implementation steps followed may also be useful to researchers that want to introduce and test other job scheduling algorithms in gLite, something that we also plan to do in our future work.

The remainder of the paper is organized as follows. In Section 2 we present a summary of the architecture of the gLite middleware. In Section 3 we describe the SFETE scheduling algorithm. In Section 4 we describe our implementation of SFETE in gLite. In Section 5 we present the experiments performed and the corresponding results. Finally, in Section 6 we conclude the paper.

2 The gLite Middleware

2.1 General

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 embedded into a consistent and secure framework. The gLite middleware runs over the Scientific Linux [15] platform and it is written mainly in C++. The gLite Grid services, which follow a Service Oriented Architecture (SOA), can be grouped into five service groups: Access Services, Security Services, Information and Monitoring Services, Data Services and Job Management Services.

A user can access the functionalities offered by the gLite middleware through the User Interface (UI) [19]. Security services [19] include the Authentication, Authorization, and Auditing operations. The users of the Grid infrastructure are divided into Virtual Organizations (VOs), which are abstract entities that group together users, institutions and resources into administrative domains. Information and Monitoring Services [19] provide a mechanism to publish and consume information on the Grid resources and their status. This information, used also for monitoring and accounting purposes, is essential for the operation of the whole Grid. In gLite there are two information systems: the Globus Monitoring and Discovery Service (MDS) [16], used for resource discovery and for publishing the resource status, and the Relational Grid Monitoring Architecture (RGMA) [17], used for accounting, monitoring and publication of user-level information. The three main services that relate to data are the Storage Element, the File & Replica Catalog Service and the Data Management [19]. The Storage Element (SE) provides the virtualization of a storage resource, which can vary from simple disk servers to complex hierarchical tape storage systems. Finally, the main Job Management Services [19] are the Computing Elements (CE) and the Workload Management System (WMS). The CE provides the virtualization of a computation resource (e.g., cluster, supercomputers or individual workstations). A CE provides information on the underlying resource and offers a common interface for submitting and managing jobs on the resource. Also, a CE has a collection of Worker Nodes (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 to the available WNs. gLite supports the following LRMS: OpenPBS/PBSPro, LSF, Maui/Torque, BQS and Condor. The Workload Management System (WMS) is a Grid level meta-scheduler that schedules jobs on the available CEs according to user preferences and several policies (see Section 2.3). It also keeps track of the jobs it manages in a consistent way, via the Logging and Bookkeeping (LB) service.

2.2 Monitoring and Discovery Service (MDS)

Figure 1 shows the Monitoring and Discovery Service (MDS) architecture. Computing and storage resources at a site run a piece of software called Information Provider, which generates the resource-related information (both static, such as the type of SE, and dynamic, such as the used space in a SE). This information is published via an LDAP server, called Grid Resource Information Server (GRIS) that normally runs on the resource itself. At each site, the Berkeley Database Information Index (BDII) is used to store and publish data from the local GRISes. In the gLite context, by the term site we mean a collection of CEs and SEs located at the same place. At the top level of the information hierarchy there are also BDIIs that collect information from the site BDIIs. Therefore the top level BDIIs contain all the available information on the Grid sites they look at. Nevertheless, it is always possible to get information about specific resources by directly contacting the GRISes.

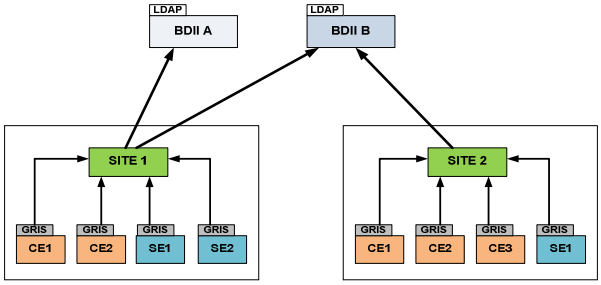


Figure 1. The architecture of the Monitoring and Discovery Service.

R-GMA is an implementation of the Grid Monitoring Architecture (GMA) [17] proposed by the Global Grid Forum (GGF) [18]. In R-GMA, information is presented as though it is stored in a global distributed relational database. This model is more powerful than the LDAP-based one, since relational databases support more and more advanced query operations.

2.3 Workload Management System (WMS)

2.3.1 WMS Architecture

A user can access the job management services, provided by the WMS, through the Workload Manager Proxy (WMProxy), using a set of tools called WMS-UI tools. The Workload Manager (WM) is the core component of the WMS and is responsible for selecting the most appropriate CE for the job's execution, taking into account the job's requirements and preferences. The decision on the resources to be used is the outcome of a matchmaking process between the job request and the available resources., Three main components of the WMS service are involved in the operation of WM: the Matchmaker (MM), the Information Super Market (ISM) and the Task Queue (TQ). The Matchmaker (MM) or Resource Broker (RB), provides a matchmaking service, which selects the resources that best match the job's requirements. The Information Super Market (ISM) is a repository of resources information that is updated at periodic intervals. 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 third fundamental component of the WM is the Task Queue (TQ) that queues a job request if no resources that match the job requirements are immediately available. The remaining of the WMS components handle the jobs during the rest of their lifetime, after the WMS has found suitable CEs for their execution.

2.3.2 Job Description File

When a user submits a job to the Grid through gLite, she also creates a job description file containing the job's characteristics and requirements. This file's syntax is defined through the Job Description Language (JDL). In the JDL file, a set of predefined attributes have a special meaning for the WMS. These attributes are usually decomposed in three categories:

- ***Job attributes***: through which the job's specific characteristics are specified.
- ***Data attributes***: through which the job's input data and SE related information are specified.
- ***Requirements and Rank***: through which the job's CE requirements and preferences are specified.

2.3.3 WMS scheduling algorithms

The Workload Manager (WM) takes as input the job's JDL file and performs two operations:

- the filtering operation, through which a set of candidate CEs is created, based on the job's requirements as these are defined from 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.

Looking in the code of gLite we identified two matchmaking algorithms, which we call the Max Rank and the Fuzzy Rank. The Max Rank algorithm chooses the CE whose current state maximizes the expression at the Rank attribute. When there are more than one CEs that have the same maximum value for the Rank attribute, 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 some kind of 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 higher for higher ranking values. Based on these probabilities the algorithm selects randomly a CE. 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.

3 Simple Fair Execution Time Estimation scheduling algorithm

The SFETE algorithm that we implemented and integrated in gLite, assigns job *i* to the resource *j* that provides the minimum *simple* fair execution time $\hat{X}_{ij}$. The simple fair execution time 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, however, taking into account the fair execution times of the other jobs assigned to the resource.

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

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

where 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. This way the fair share of the computation capacity of the jobs assigned to the resource changes, but their simple fair execution time estimations are not re-estimated. Note that the FETE algorithm introduced in [9] takes new arrivals and job completions into account in estimating job fair execution times, but is considerably more complicated than SFETE without having appreciably better performance, as simulation results indicated. Therefore, SFETE can be viewed as a good approximation to FETE, which obtains most of its performance and fairness benefits, at a fraction of its implementation cost. More importantly, SFETE has no need for the a-priori knowledge of the job workloads, as FETE does.

4 Implementation

4.1 The ETICS system

In order for a developer to be able to extend and build the components of gLite, the ETICS system [20] is needed. ETICS, which stands for eInfrastructure for Testing, Integration and Configuration of Software, is an on-line 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 (*etics-checkout command*) the necessary modules, files, libraries, configuration files needed to build the component of interest (e.g., the WMS). Once the code and a suitable configuration have been checked out, the developer can use the ETICS client to generate software packages and reports (*etics-build*).

4.2 Max and Fuzzy implementation at gLite

To implement SFETE in gLite, we first studied the code structure of the Workload Management System (WMS) and especially the way Max and Fuzzy Rank are implemented. The implementation of the main process of the WMS, that is of the Workload Manager (WM), is located in directory *org.glite.wms.manager/src /daemons*. The WM reads the configuration files, initializes various WMS services (e.g., the Helper, the Task Queue, the ISM) and starts a number of concurrent threads for serving multiple job requests. The Helper service, located in the directory *org.glite.wms.helper/src/*, is the interface of the WM with the various components of the WMS, though which the management of the job's execution cycle is performed. The most important file of the Helper service is the *org.glite.wms.helper/src/broker/ Helper_ism.cpp.* In this file the Helper service informs the WM for the filtering and selection strategies the user defined in the job's JDL file. For example, when the user thinks that a job has increased data transfer requirements, then she can define a filtering strategy, which selects the CEs that are closer to the required SEs. This way the data transfers in the network are reduced. Moreover, if the expression "*FuzzyRank=true"* is used, the Helper informs the WM that it must use the Fuzzy Rank selection strategy.

The implemented filtering strategies are located in the *org.glite.wms.broker/src* directory, while the selection strategies are in the *org.glite.wms.broker/src/selector* directory. The default filtering strategy is implemented in the *RBSimpleISMImpl.cpp* file. The selection strategies are first defined in the *RBSelectionSchema.cpp* file and then implemented in separate files. As already mentioned, gLite implements the Max Rank (*maxRankSelector.cpp*) and the Fuzzy Rank (*stochasticRankSelector.cpp*) selection strategies. Both the filtering and the selection strategies use the functions of the Matchmaker (MM), which is another part of the WMS. The implementation of the MM is located in the *matchmakerISMImpl.cpp* file under the directory *org.glite.wms.matchmaking/src.* MM performs the matching between the user requirements and the available CEs, using the job's JDL file and the information provided by the ISM service. Also, the MM calculates the Rank expression of each CE in the set of CEs created by the filtering strategy. Figure 2 summarizes the structure of the gLite's directories where the WMS code is located.

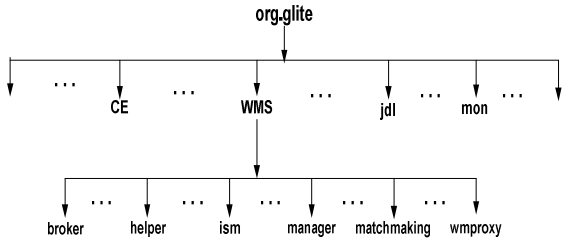


Figure 2. The structure of the gLite's directories where the WMS code is located.

4.3 SFETE implementation in gLite middleware

SFETE uses the two-phase procedure (filtering and selection) described above, in order 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. So, 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. In order for a user to be able to use SFETE algorithm, we defined a new boolean JDL attribute, called *FeteRank* (Figure 3).

```
[
 Type="Job";
 JobType="Normal";
 Executable="fibonacci.sh";
 StdOutput="fibonacci.out";
 StdError="fibonacci.err";
 InputSandbox={"./fibonacci.sh","./Fibonacci.py"};
 OutputSandbox={"fibonacci.out", "fibonacci.err"};
 Requirements=other.GlueCEStateWaitingJobs<100;
 FeteRank=true;
]
```

Figure 3. A job's JDL file requesting the use of the SFETE selection strategy. The Rank attribute is not needed.

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 the jobs assigned to it. In order to obtain the computation capacity information, SFETE asks the Information Super Market (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 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. The Monitoring and Discovery Service (MDS) service updates this information in the ISM at periodic intervals. 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. The basic idea of the mechanism is the use of a local counter for each CE, which is updated from the WM every time a job is assigned for execution to the corresponding CE. This value is of course only an estimate, since 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. For this reason in every ISM update, the local counters are updated with the data published from the ISM.

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.cc*).
- **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 use the *SFeteRankSelector* 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 mechanism that keeps track of the jobs that have been assigned to the available CEs.

5 Performance Results

5.1 Testbed

In order to evaluate the performance of the SFETE scheduling algorithm 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 4 shows the Grid infrastructure we set up, which was based both on gLite 3.0 and gLite 3.1 middleware, since at the time of we set it up, the gLite 3.1 was not fully released yet.

Our testbed has three CEs, each of which has its own WN (Table 1), for executing the jobs. The CEs use as LRMS the Torque/PBS system. The R-GMA is responsible for publishing information about the available computation and storage resources and the SE provides access to the data storage resources. The LFC catalog service stores the location or locations of the files and their replicas. The R-GMA, the SE and the LFC components where not utilized in our experiments. The Site-BDII informs the ISM component of the WMS about the status of the various CEs. The WMS we use is modified as we have added to it the SFETE scheduling algorithm. VOMS provides information such as VO membership, group membership and the roles of each user. In our experiments we use a single VO that gets 100% of each CE's computational capacity. Finally, through the User Interface (UI) each user can be authenticated and authorized to use the Grid resources. In order for authentication to work within gLite all users and services need to have a certificate issued from a trusted Certificate Authority (CA). Since the CA is the heart of the gLite authentication system, we established for simplicity our own trusted CA in our Grid testbed.

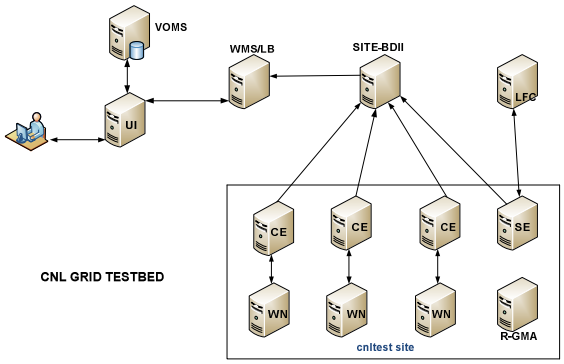


Figure 4. The gLite testbed used in our experiments.

Resource Type	Details
SE	Pentium 4 1.6 GHz Hard Disk 60GB
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

Table 1: The characteristics of the storage and computational resources of our testbed.

5.2 Experiment Parameters

In order to evaluate the performance of the SFETE against the default gLite scheduling algorithms, we performed a number of experiments. 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. Based on the work we performed in [10], we observed that in the EGEE infrastructure job inter-arrival times follow an exponential distribution and therefore Poisson is a good approximation of the arrival process. We do not use larger arrival rates since our Grid testbed has limited computation resources and larger rates would overload the testbed. Users generate jobs of different mean durations: 5, 6.5, 8.5, 10.5 and 12.5 minutes. 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 do not have any data dependencies, since this would complicate our study.

In our experiments we use a default filtering strategy, by setting the Requirement attribute 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 algorithms, and not those of SFETE. In our experiments we used the following metrics:

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

5.3 Results

Figure 5 shows the average job total delay as a function of the job submission rate. For all the scheduling algorithms examined when the job submission rate increases the average job total delay increases. 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 6.

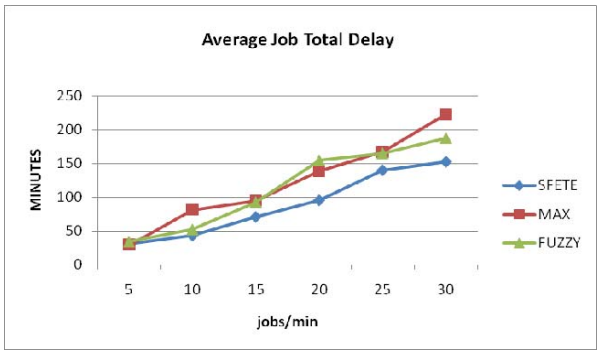
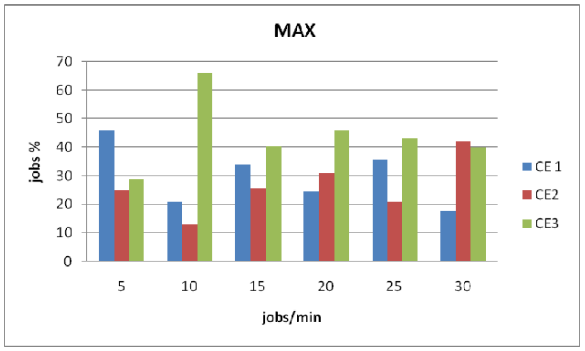


Figure 5. The average job total delay for the Max Rank, the Fuzzy Rank and the SFETE algorithms, for various job submission rates.

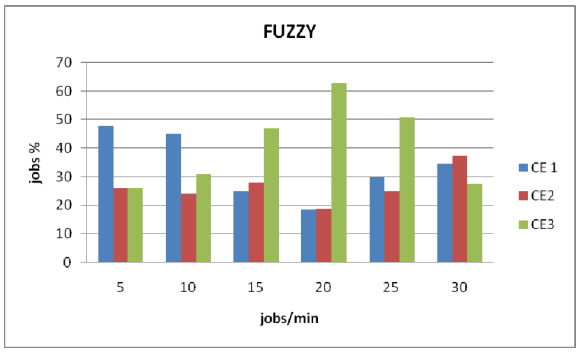
Figure 6 shows that the SFETE scheduling algorithm results in constant distribution of the jobs to the available computational resources, independently of the job submission rate. SFETE assigns each job to the computational resource that provides, at the time of the submission, the minimum simple fair execution time estimation. The result of this process is independent from the job submission rate, since SFETE assigns to each available resource a percentage of the total jobs, 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 a chance that a resource is 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 in order to evaluate the Rank expression at the job's JDL file. 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

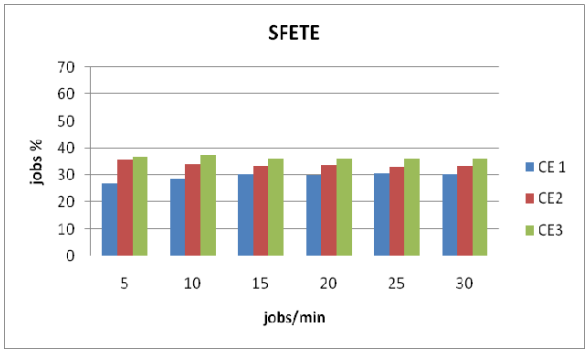
that the Max and Fuzzy Rank take are based on outdated information and they may be incorrect. The effects of outdated information become even more pronounced as the job submission rate increases. On the other hand, the mechanism we implemented in SFETE provides a better estimate of the number of jobs 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.



(a)



(b)



(c)

Figure 6. The job percentage distributions at the available CEs that (a) Max Rank, (b) Fuzzy Rank and (c) SFETE achieve, for various job submission rates.

Figure 7 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, since it better distributes the jobs at the available resources.

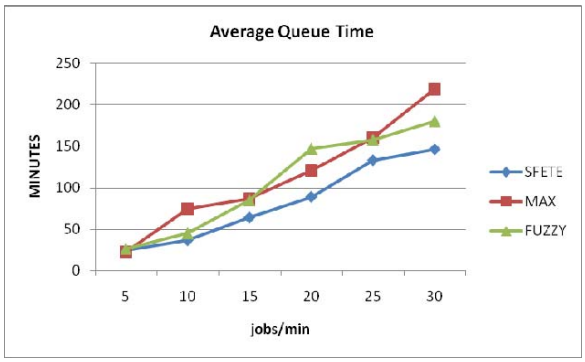


Figure 7. The average time jobs remain at the queues of the CEs for the Max Rank, the Fuzzy Rank and the SFETE algorithms, for various job submission rates.

6 Conclusions

We have investigated the gLite middleware architecture and especially the part related to the scheduling algorithms provided by default to its users. Moreover, we developed and integrated in gLite a new scheduling algorithm. The performance of this algorithm was evaluated experimentally in a small Grid testbed, and was shown to outperform the existing gLite scheduling algorithms. The main drawback of the current implementation of the Max and the Fuzzy Rank algorithms is the outdated information that the gLite's information service publishes, for a number of dynamic characteristics, such as the number of jobs at the resource queues. Our results indicate that simple changes in the gLite's scheduling procedures can yield significant performance gains. In the future we also plan to consider the case where each VO gets a percentage of the CE's capacity. Finally, given the absence of any information on the web about the gLite's architecture and code structure, we believe that this work is a good starting point for anyone interested in extending, improving and evaluating the gLite's mechanisms.

7 References

[1] T. Braun et al, A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, J. of Parallel and Distributed Computing, Vol. 61, No. 6, pp. 810-837, 2001.

[2] H. Dail, H. Casanova and F. Berman, A Decoupled Scheduling Approach for Grid Application Development Environments, J. of Parallel and Distributed Computing, Vol. 63, pp. 505-524, 2002.

[3] Y. Cardinale, H. Casanova, An evaluation of Job Scheduling Strategies for Divisible Loads on Grid Platforms, HPC&S, 2006.

[4] R. Buyya, M. Murshed, D. Abramson, S. Venugopal, Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm, Intl J. of Software: Practice and Experience (SPE), Vol. 35, No. 5, pp. 491-512, 2005.

[5] K. H. Kim, R. Buyya, Fair Resource Sharing in Hierarchical Virtual Organizations for Global Grids, Intl Conf. on Grid Computing, 2007.

[6] N. Doulamis, E. Varvarigos, T. Varvarigou, Fair Scheduling Algorithms in Grids, Tran. on Parallel and Distributed Systems, Vol. 18, No. 11, pp. 1630-1648, 2007.

[7] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Software: Practice and Experience, Vol. 32, No. 2, pp. 135-164, 2002.

[8] A. Kertesz, P. Kacsuk, Distributed and Parallel Systems: A Taxonomy of Grid Resource Brokers, Springer, 2007.

[9] H. Dafouli, P. Kokkinos, E. Varvarigos, Fair Execution Time Estimation Scheduling in Computational Grids, International Conference on Distributed and Parallel Systems (DAPSYS), pp. 93-104, 2008.

[10] K. Christodoulopoulos, V. Gkamas, E. Varvarigos, Statistical Analysis and Modeling of Jobs in a Grid Environment, Journal of Grid Computing, Vol. 6, No. 1, pp. 77-101, 2007.

[11] www.glite.org

[12] http://www.eu-egee.org/

[13] http://www.globus.org/

[14] http://www.unicore.eu/

[15] https://www.scientificlinux.org/

[16] MDS 2.2 Features in the Globus Toolkit 2.2 Release: http://www.globus.org/toolkit/mds/#mds gt2

[17] R-GMA: Relational Grid Monitoring Architecture: http://www.r-gma.org/index.html

[18] B. Tierney et al. , A Grid Monitoring Architecture, GGF , 2001 (revised 2002)

[19] gLite 3.1 User Guide Manuals Series: https://edms.cern.ch/file/722398//gLite-3-UserGuide.pdf

[20] http://etics.web.cern.ch/etics/