

Adaptive scheduling solution for grid meta-brokering*

Attila Kertész[†], József Dániel Dombi[‡] and József Dombi[‡]

Abstract

The nearly optimal, interoperable utilization of various grid resources play an important role in the world of grids. Though well-designed, evaluated and widely used resource brokers have been developed, these existing solutions still cannot cope with the high uncertainty ruling current grid systems. To ease the simultaneous utilization of different middleware systems, researchers need to revise current solutions. In this paper we propose advanced scheduling techniques with a weighted fitness function for an adaptive Meta-Brokering Grid Service, which enables a higher level utilization of the existing grid brokers. We also set up a grid simulation environment to demonstrate the efficiency of the proposed meta-level scheduling solution. The presented evaluation results show that the proposed novel scheduling technique in the meta-brokering context delivers better performance.

Keywords: Grid Computing, Meta-Brokering, Scheduling, Grid Service, Random Number Generator function

1 Introduction

Ten years ago a new computing infrastructure called the Grid was born. Ian Foster et. al. made this technology immortal by publishing the bible of the Grid [1] in 1998. Grid Computing has become an independent research field since then: currently grids are targeted by many world-wide projects. A decade is a long time: though the initial goal of grids to serve various scientific communities by providing a robust hardware and software environment is still unchanged, different middleware solutions have been developed (Globus Toolkit [2], EGEE [3], UNICORE [4], etc.). The realizations of these grid middleware systems formed production grids that are mature enough to serve scientists having computation and data intensive applications. Nowadays research directions are focusing on user needs: more efficient utilization and interoperability play the key roles. To solve these problems

*This work was supported by the FP7 Network of Excellence S-Cube funded by the European Commission (Contract FP7/2007-2013).

[†]MTA SZTAKI, University of Szeged E-mail: keratt@inf.u-szeged.hu

[‡]University of Szeged E-mail: {dombijd,dombi}@inf.u-szeged.hu

grid researchers have two options: as a member of a middleware developer group they can come up with new ideas or newly identified requirements and go through the long process of designing, standardizing and implementing the new feature, then waiting for the next release containing the solution. Researchers sitting on the other side or unwilling to wait for years for the new release, need to rely on the currently available interfaces of the middleware components and use advanced techniques of other related research fields (peer-to-peer, web computing, artificial intelligence, etc.). We have chosen the second option to improve grid resource utilization with an interoperable resource management service.

Since the management and advantageous utilization of highly dynamic grid resources cannot be handled by the users themselves, various grid resource management tools have been developed, supporting different grids. User requirements created certain properties that resource managers have learned to support. This development is still continuing, and users already need to stress themselves to distinguish brokers and to migrate their applications, when they move to a different grid. Interoperability problems and multi-broker utilization have emerged the need for higher level brokering solutions. The meta-brokering approach means a higher level resource management by enabling automatic and simultaneous utilization of grid brokers. Scheduling at this level requires sophisticated approaches, because high uncertainty presents at all stages of grid resource management. Despite these difficulties, this work addresses the resource management layer of middleware systems and proposes an enhanced scheduling technique to improve grid utilization in a high-level brokering service.

In the following sections of this paper we are focusing on a meta-brokering solution for grid resource management and present an adaptive scheduling technique that targets better scheduling in global grids. In Section 2 we introduce meta-brokering in grids, in Section 3 we describe our proposed scheduling solution, and in Section 4 we present our simulation architecture and the evaluation of our proposed solution. Finally, in Section 5 we gather the related research directions and Section 6 concludes the paper.

2 The need for grid meta-brokering

Heterogeneity appears not only in the fabric layer of grids, but also in the middleware. Even components and services of the same middleware may support different ways for accessing them. After a point this variety slows down grid development, and makes grid systems unmanageable. Most grid middleware systems have fixed interfaces to access their components and propagate information flow. In case of resource management most of the middleware systems provide access to static properties (number of CPUs, size of memory, etc.) and some also give dynamic ones (number of waiting jobs, expected response time), but this data is usually outdated due to timely periodic refreshing. As a result we can state that there is a high uncertainty in current grids, which is not likely to change soon. Though the first de facto middleware, the Globus Toolkit [2], did not have a resource broker that au-

tomates resource selection, the currently used middleware systems have built-in or supporting brokers [8]. The development of different brokers and grids has started a separation process in the research and user community, too. Therefore one of the major problems of current grids is grid interoperability. Focusing on the resource management layer of grids an obvious solution would be to interconnect brokers to create interoperability. Unfortunately current brokers do not have a common protocol and uniform interface for intercommunication, though the OGF-GSA [10] started to work on this issue. Once they standardize a solution we still would need to wait till all the brokers implement it in order to establish interoperability. In order to achieve this goal in a short term we have chosen to interconnect brokers by a high-level resource manager: we introduced meta-brokering (first proposed in [7]) that means a higher level utilization of the existing, widely used and reliable resource brokers. Since most of the users have certificates to access more Grids, they are facing the problem of grid selection: which grid, which broker should I choose for my specific application? Just like users needed resource brokers to choose proper resources within a grid, now they need a meta-brokering service to decide, which broker (or grid) is the best for them and also to hide the differences of utilizing them. In this way the meta-brokering approach solves the grid interoperability problem at the level of resource management by providing a uniform interface for the users of all the grids they have access to.

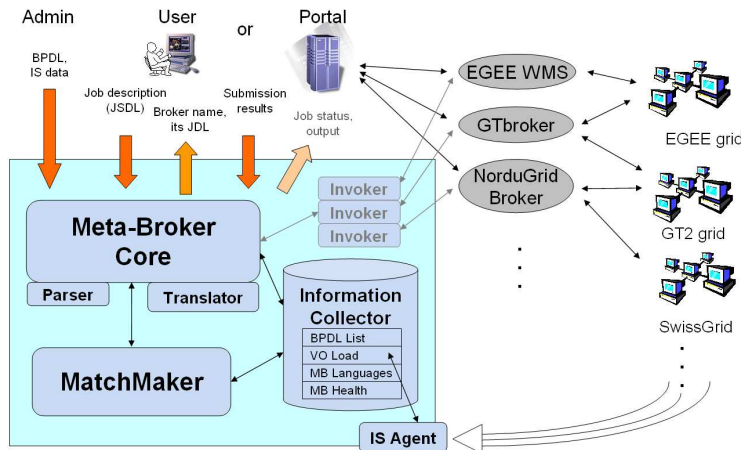


Figure 1: Components of the Grid Meta-Broker Service

Figure 1 introduces the revised architecture of the Grid Meta-Broker Service that enables the users to access resources of different grids through their own brokers. In this way, this higher level tool matches resource brokers to user requests. The system is implemented as a web-service that is independent from middleware-specific components. The provided services can be reached through WSDL (Web Services Description Language). In the following we give a brief summary of its com-

ponents and their operation. As the JSDL (Job Submission Description Language) standard [5] proposed by OGF (Open Grid Forum [9]) is general enough to describe jobs of different grids and brokers, we have chosen this to be the job description language of the Meta-Broker. The Translator components of the Meta-Broker are responsible for transforming the resource specification defined by the user to the language of the appropriate resource broker that the Meta-Broker selects to use for a given job. Regarding all the various job specification formats used by different grid middleware systems not all job attributes can be expressed in each document. Furthermore we revealed some useful scheduling-related attributes that are also missing from JSDL. To overcome these limitations we specified MBSDL (Meta-Broker Scheduling Description Language [6]). The main attribute categories are: middleware constraints, scheduling policies and QoS requirements. This schema can be used to extend JSDL with scheduling-related attributes. Besides describing user jobs, we also need to describe resource brokers in order to differentiate and manage them. These brokers have various features for supporting different user needs. These needs should be able to be expressed in the users JSDL, and identified by the Meta-Broker for each corresponding broker. Therefore we proposed an extendable BPDFL (Broker Property Description Language [6]) – similar to the purpose of JSDL –, to express metadata about brokers. The common subset of the individual broker properties is stored here: the supported middleware, job types, certificates, job descriptions, interfaces, monitoring features and dynamic performance data. The scheduling-related ones are stored in MBSDL: fault tolerant features (checkpointing, rescheduling, replication), agreement support, scheduling policies (ranking by resource attributes) and QoS properties (e.g. advance reservation, co-allocation, email notification). The union of these properties forms a complete broker description document that can be filled out and regularly updated for each utilized resource broker. These two kinds of data formats are used by the Meta-Broker: JSDL is used by the users to specify jobs and the BPDFL (Broker Property Description Language) by administrators to specify brokers – both parties can use MBSDL to extend their descriptions.

The Information Collector (IC) component of the Meta-broker stores the data of the reachable brokers and historical data of the previous submissions. This information shows whether the chosen broker is available, or how reliable its services are. During broker utilization the successful submissions and failures are tracked, and regarding these events a rank is modified for each special attribute in the BPDFL of the appropriate broker (these attributes were listed above). In this way, the BPDFL documents represent and store the dynamic states of the brokers. All data is stored in XML, and advanced XML-serialization techniques are used by the IC. The load of the resources behind the brokers is also taken into account to help the Matchmaker to select the proper environment for the actual job. When too many similar jobs are needed to be handled by the Meta-Broker an eager matchmaking may flood a broker and its grid. That is the main reason why load balancing is an important issue. In order to cope with this problem, there is an IS (Information System) Agent component reporting to the Information Collector, which regularly checks the load of the underlying grids of each connected resource broker, and store this data. This

tool is implemented as separate web-service connected to the Information System of the grids behind the utilized brokers. With the additional information provided by this agent the matchmaking process can adapt to the load of the utilized grids. Finally, the actual state (load, configurations) of the Meta-Broker is also stored here, and it can also be queried by users. The continuous monitoring of grid load and broker performances makes this grid service self-adaptive.

The previously introduced languages are used for matching the user requests to the description of the interconnected brokers: which is the role of the Matchmaker component. The JSDL contains the user request (this supposed to be an exact specification of the user's job) using the extended attributes, while the interconnected brokers are described by their BPDFL documents. The default matchmaking process consists of the following steps to find the fittest broker:

- The Matchmaker compares the JSDL of the actual job to the BPDFL of the registered resource brokers. First the job requirement attributes are matched against the broker properties stored in their BPDFLs: this selection determines a group of brokers that are able to submit the job. This phase consists of two steps: first the brokers are filtered by all the requirements stated in the JSDL. When none of the brokers can fulfill the request, another filtering process will be started with minimal requirements (those ones are kept which are real necessary for job execution). If the available brokers still can not accept the job, it will be rejected.
- In the second phase the previous submissions of the brokers and the load of the underlying grids are taken into account: The MatchMaker counts a rank for each of the remaining brokers. This rank is calculated from the load that the IS Agent regularly updates, and from the job completion rate that is updated in the PerformanceMetrics field of the BPDFL for each broker. When all the ranks are counted, the list of the brokers is ordered by these ranks.
- Finally the first broker of the priority list is selected for submission.

3 Adaptive Scheduling for meta-brokering

In the previous section we introduced the Grid Meta-Broker and shown how the default matchmaking is carried out. The main goal of this paper is to enhance the scheduling part of this matchmaking process. To achieve this, we have created a Decision Maker component and inserted it into the MatchMaker component of the Meta-Broker (see Figure 1). The first part of the matchmaking is unchanged: the list of the available brokers is filtered according to the requirements of the actual job read from its JSDL. Then the list of the remaining brokers along with their performance data and background grid load are sent to the Decision Maker in order to determine the fittest broker for the actual job. The scheduling techniques and the process are described in the following paragraphs.

The Decision Maker uses a random number generator, and we chose a JAVA solution, which generates pseudorandom numbers. This generator produces exactly

the same sequence of random numbers for each execution with the same initial value. This initial value is called the seed. The JAVA random number generator class uses uniform distribution and 48-bit seed, which is modified by a linear congruential formula[11]. The default seed is the current time in milliseconds since 1970. We also developed a unique random number generator, which generates random numbers with a given distribution. We called this algorithm as generator function. In our case we defined a score value for each broker, and we created the distribution based on the score value. For example the broker which has the highest score number has the highest probability to be chosen. In this algorithm the inputs are the broker id and the broker score, which are integer numbers (see Table 1).

Table 1: Inputs of the algorithm

BrokerID	Score
3	2
4	3
5	1
6	2

The next step is to choose a broker and put it into a temporary array: the cardinality is determined by the score value (see Table 2).

Table 2: Elements in the temporary array

Broker ID	3	3	4	4	4	5	6	6
Array ID	1	2	3	4	5	6	7	8

After the temporary array is filled, we shuffle this array and choose an array element using the JAVA random generator. In the example shown in Table 3 the generator function chose the broker with id 4.

Table 3: Shuffled temporary array

Broker ID	4	3	6	3	4	4	5	6
Array ID	1	2	3	4	5	6	7	8

Java Random generator: **5**

To improve the scheduling performance of the Meta-Broker we need to send the job to the broker that best fits the requirements and executes the job without failures with the shortest execution time. Every broker has three properties that

the algorithm can rely on: the successful counter, the failure counter and the load counter.

- The successful counter represents the number of jobs which had finished without any errors.
- The failure counter shows the number of failed jobs.
- The load counter indicates the actual load of the grid behind the broker (in percentage).

We have developed four different kinds of decision algorithms. The trivial algorithm uses only a random number generator to select a broker. The other three algorithms take into account the previously mentioned broker properties. These algorithms define a score number for each broker and use the generator function to select one. To calculate the score value we build a weighted sum of the evaluated properties. This number is always an integer number. Furthermore, the second and third decision algorithms take into account the maximum value of the failure and load counter. This means that we extract the maximum value of the properties before multiplying them with the weight. The generator function of the third algorithm chooses a broker which score number is not smaller than the half of the highest score value.

After testing different kinds of weighted systems, we conclude that the most useful weights are shown in Table 4) that represent the weights of the used decision algorithms.

Table 4: The weights of the decision makers

Decision Maker	Success_weight	Failed_weight	Load_weight
Decision I.	3	0.5	1
Decision II.	4	4	4
Decision III.	4	4	4

During the utilization of the Meta-Broker, the first two broker properties (successful and failure counter) are incremented through a feedback method that the simulator (or a user or portal in real world cases) calls after the job submission is finished. The third property, the load value, is queried by the Meta-Broker from an information provider (Information System) of a Grid. During simulation this data is saved to a database by the Broker entities of the simulator (described later and shown in Figure 2). This means by the time we start the evaluation and till we do not receive feedback from finished jobs the algorithms can only rely on the background load of the grids. To further enhance the scheduling we developed a training process that can be executed before the simulation in order to initialize the first and second properties. This process sends a small number of jobs with various properties to the brokers and set the successful and failed jobs number at

the BPDs of the brokers. With this additional training method we expect shorter execution times by selecting more reliably brokers.

4 Evaluation

In order to evaluate our proposed scheduling solution, we have created a general simulation environment, in which all the related grid resource management entities can be simulated and coordinated. The GridSim toolkit [12] is a fully extendable, widely used and accepted grid simulation tool – these are the main reasons why we have chosen this toolkit for our simulations. It can be used for evaluating VO-based resource allocation, workflow scheduling, and dynamic resource provisioning techniques in global grids. It supports modeling and simulation of heterogeneous grid resources, users, applications, brokers and schedulers in a grid computing environment. It provides primitives for the creation of jobs (called gridlets), mapping these jobs to resources and their management, therefore resource schedulers can be simulated to study scheduling algorithms. GridSim provides a multilayered design architecture based on SimJava [13], a general purpose discrete-event simulation package implemented in Java. It is used for handling the interaction or events among GridSim components. All components in GridSim communicate with each other through message passing operations defined by SimJava.

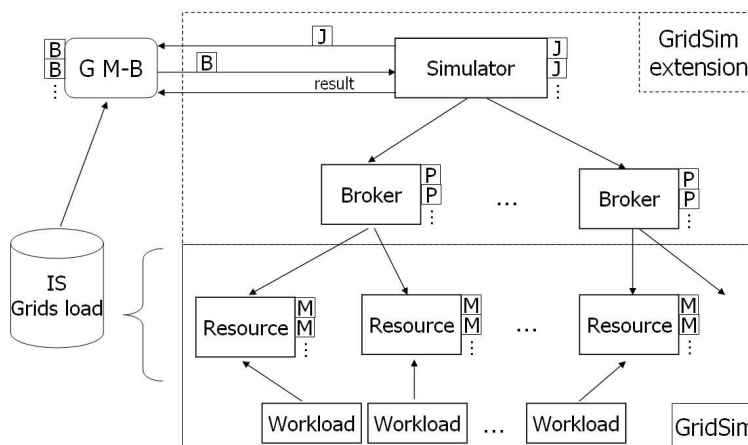


Figure 2: Meta-Broking simulation environment based on GridSim

Our general simulation architecture can be seen in Figure 2. On the bottom-right part we can see the GridSim components used for the simulated grid systems. Resources can be defined with different grid-types. Resources consist of more machines, to which workloads can be set. On top of this simulated grid infrastructure we can set up brokers. The Broker and Simulator entities have been developed by us in order to enable the simulation of meta-broking. Brokers are extended

GridUser entities:

- they can be connected to one or more resources;
- different properties can be set to these brokers (agreement handling, co-allocation, advance reservation, etc.);
- some properties can be marked as unreliable;
- various scheduling policies can be defined (pre-defined ones: rnd – random resource selection, fcpu – resources having more free cpus or less waiting jobs are selected, nfailed – resources having less machine failures are selected);
- generally resubmission is used, when a job fails due to resource failure;
- finally they report to the IS (Information System) Grid load database by calling the feedback method of the Meta-Broker with the results of the job submissions (this database has a similar purpose as a grid Information System).

The Simulator is an extended GridSim entity:

- it can generate a requested number of gridlets (jobs) with different properties, start and run time (length);
- it is connected to the created brokers and able to submit jobs to them;
- the default job distribution is the random broker selection (though at least the middleware types are taken into account);
- in case of job failures a different broker is selected for the actual job;
- it is also connected to the Grid Meta-Broker through its web service interface and able to call its matchmaking service for broker selection.

4.1 Preliminary testing phase

Table 5 shows the details of the preliminary evaluation environment. 10 brokers can be used in this simulation environment. The second column denotes the scheduling policies used by the brokers: fcpu means the jobs are scheduled to the resource with the most free cpus, nfail means those resources are selected that have less machine failures, and rnd means randomized resource selection. The third column shows the capabilities/properties (eg: coallocation, checkpointing, ...) of the brokers: three properties are used in this environment, subscript F means unreliability, a broker having such a property may fail to execute a job with the requested service with a probability of 0.5. The fourth column contains the number of resources utilized by a broker, while the fifth column contains the number of background jobs submitted to the broker (SDSC BLUE workload logs taken from the Parallel Workloads Archive [14]) during the evaluation timeframe.

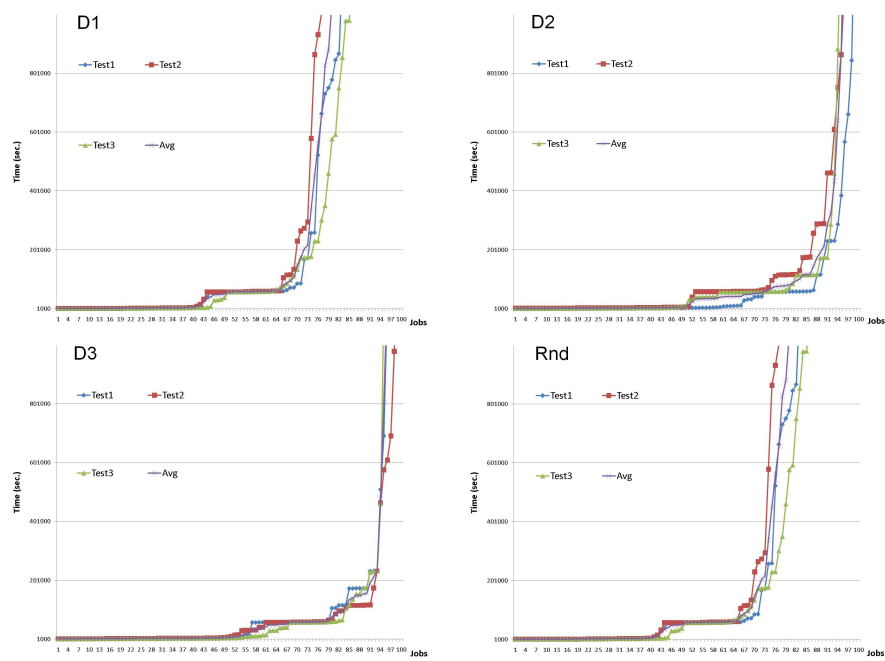


Figure 3: Diagrams of the preliminary evaluation for each algorithm

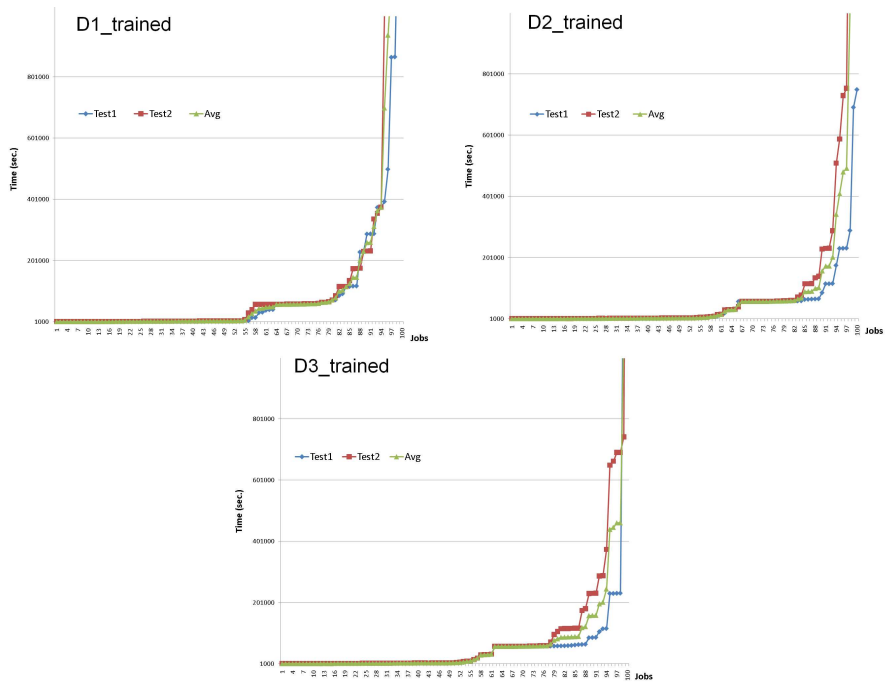


Figure 4: Diagrams of the preliminary evaluation for each algorithm with training phases

Table 5: Preliminary evaluation setup.

Broker	Scheduling	Properties	Resources	Workload
1.	fcpu	A	8	20*8
2.	fcpu	B	8	20*8
3.	fcpu	C	8	20*8
4.	fcpu	A_F	8	20*8
5.	fcpu	B_F	8	20*8
6.	fcpu	C_F	8	20*8
7.	nfail	A_FB	10	20*10
8.	nfail	AC_F	10	20*10
9.	nfail	B_FC	10	20*10
10.	rnd	-	16	20*16

As shown in the table we utilized 10 brokers to execute our first experiment. In this case we submitted 100 jobs to the system, and measured the makespan of all the jobs (time elapsed from submission till the successful finishing, including waiting time in the queue of the resources and resubmissions on failures). Out of the 100 jobs 40 had no special property (this means all the brokers can successfully execute them), for the rest of the jobs the three properties were distributed equally: 20 jobs had property A, 20 had B and 20 had C. Each resource of the simulated grids has been utilized by 20 background jobs (workload) with different submission times according to the distribution defined by the SDSC BLUE workload logs.

Figure 3 shows the detailed evaluation runs with the scheduling algorithms Decision 1 (D1), 2 (D2), 3 (D3) and without the use of the Meta-Broker (randomized broker selection – Rnd) respectively. Figure 4 shows the measured values with the three algorithms with training (we submitted 10 jobs to each broker to set their initial performance values). In Figure 5 we can see the averages of the tests with the different algorithms. This illustrates best the differences of the simulations with and without the use of the Meta-Broker.

After we have seen the diagrams of the preliminary evaluations we can state that all the proposed scheduling algorithms (D1, D2 and D3) provide shorter execution times than the random broker selection. In the main evaluation phases our goal was to set up a more realistic environment and to experience with a higher number of jobs.

4.2 Main testing phase

Table 6 shows the evaluation environment used in the main evaluation. The simulation setup was derived from real-life production grids: current grids and brokers support only a few special properties: we used four. To determine the (proportional) number of resources in our simulated grids we compared the sizes of current production grids (EGEE VOs, DAS3, NGS, Grid5000, OSG, ...). We used the same

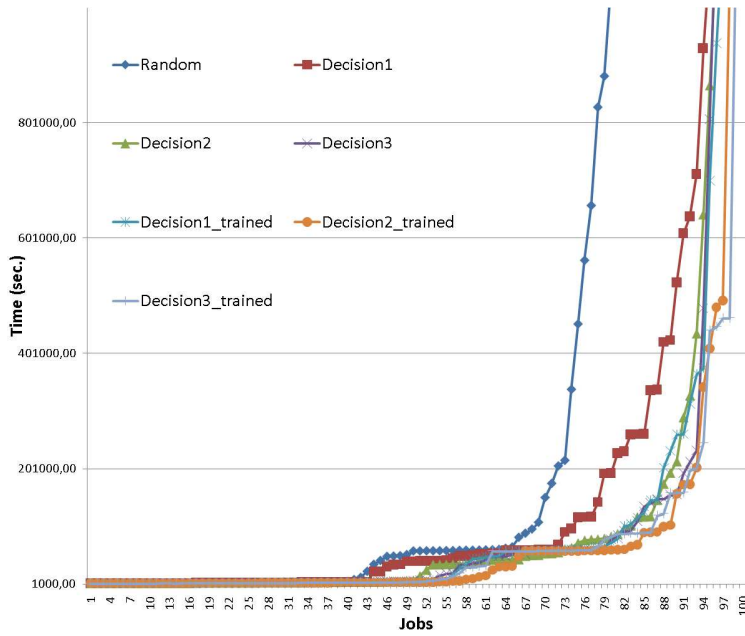


Figure 5: Summary diagram of the preliminary evaluation

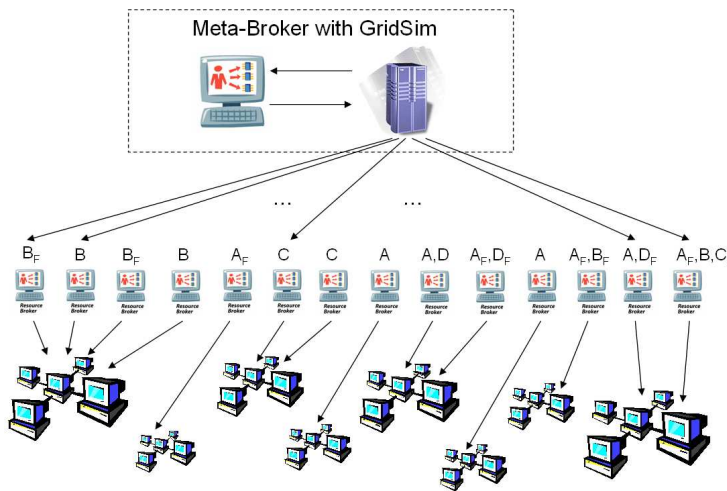


Figure 6: Simulation in the main evaluation environment

Table 6: Main evaluation setup.

Broker	Scheduling	Properties	Resources	Workload
1.	fcpu	A	6	50*6
2.	fcpu	A_F	8	50*8
3.	fcpu	A	12	50*12
4.	fcpu	B	10	50*10
5.	fcpu	B_F	10	50*10
6.	fcpu	B	12	50*12
7.	fcpu	B_F	12	50*12
8.	fcpu	C	4	50*4
9.	fcpu	C	4	50*4
10.	fcpu	A_FD	8	50*8
11.	fcpu	AD	10	50*10
12.	fcpu	AD_F	8	50*8
13.	fcpu	AB_F	6	50*6
14.	fcpu	ABC_F	10	50*10

notations in this table as before.

In the main evaluation we utilized 14 brokers. In this case we submitted 1000 jobs to the system, and again measured the makespan of all the jobs. Out of the 1000 jobs 100 had no special property, for the rest of the jobs the four properties were distributed in the following way: 30 jobs had property A, 30 had B, 20 had C and 10 had D. The workload logs contained 50 jobs for each resource. In the training processes 100 jobs were submitted to each broker prior the evaluations to set the initial values. Figure 6 shows the graphical representation of the simulation environment.

In the first phase of the main evaluation the simulator submitted all the jobs at once, just like in the preliminary evaluation. The results for this phase can be seen in Figure 7.

In the first phase we could not exploit all the features of the algorithms, because we submitted all the jobs at once and the performance data of the brokers were not updated early enough for the matchmaking process. To avoid this, in the last phase of the main evaluation we submitted the jobs periodically: 1/3 of the jobs were submitted in the beginning, then the simulator waited for 200 jobs to finish and update the performances of the brokers. After this the simulator submitted again 1/3 of all the jobs and waited for 300 more to finish. Finally the rest of the jobs (1/3 again) were submitted. In this way the broker performance results could be used by the scheduling algorithms. Figure 8 shows the results of the last evaluation phase. Here we can see that the runs with training could not make too much advantage of the trained values, because the feedback of the first submission period compensates the lack of training.

Figure 9 displays the summary of the different evaluation phases. The depicted

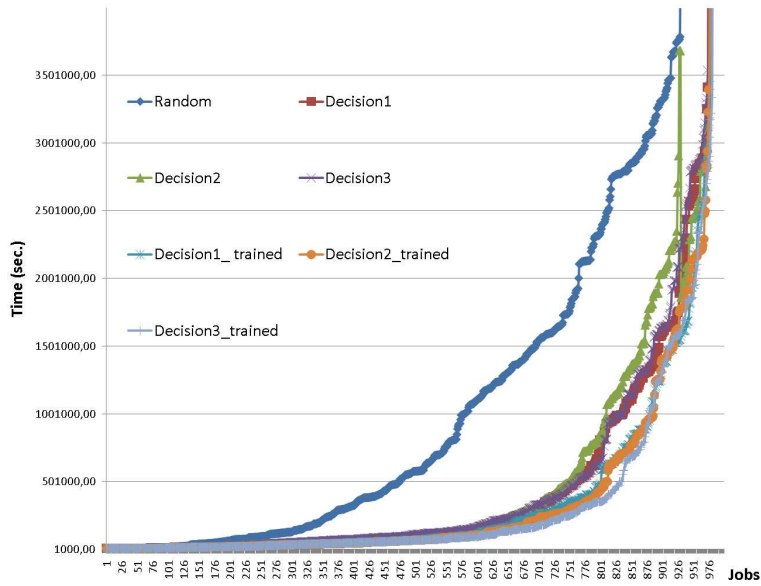


Figure 7: Diagram of the first phase of the main evaluation

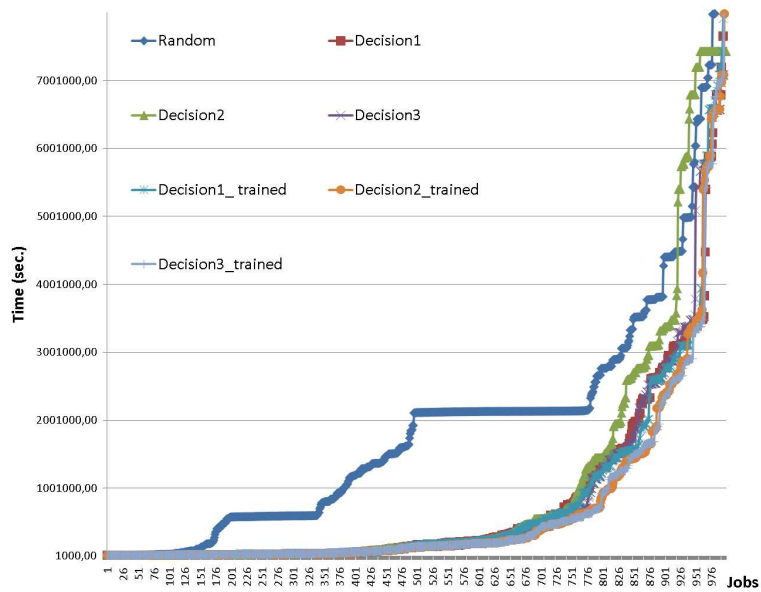


Figure 8: Diagram of the second phase of the main evaluation

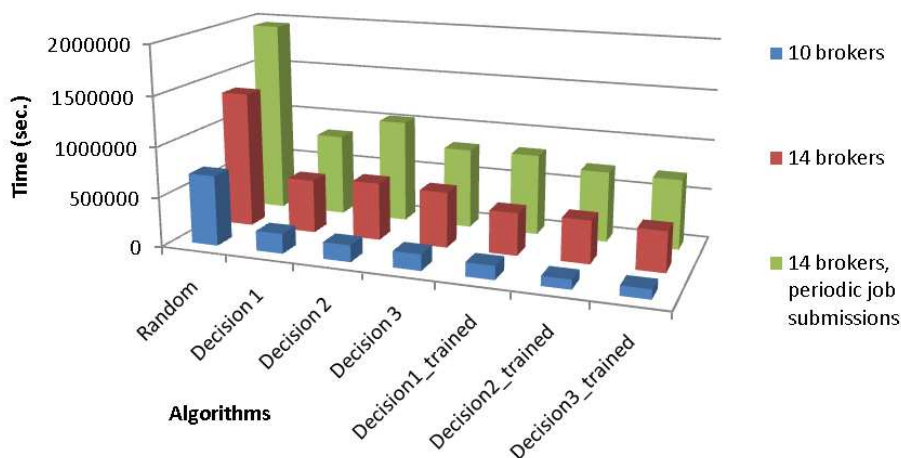


Figure 9: Summary of the evaluation results

columns show the average values of each evaluation runs with the same parameters. The results clearly show that the more intelligence (more sophisticated methods) we put into the system the higher performance we gain. The most advanced version of our proposed meta-brokering solution is the Decision Maker with the algorithm called Decision3 with training. Once the number of brokers and job properties will be high enough to set up this Grid Meta-Broker Service for inter-connecting several Grids, with the presented scheduling algorithms our service will be ready to serve thousands of users even under high uncertainty.

5 Related work

Meta-brokering means a higher level solution that brokers user requests among various grid domains. One of these promising approaches aims at enabling communication among existing resource brokers. The GSA-RG of OGF [10] is currently working on a project enabling grid scheduler interaction. They try to define common protocol and interface among schedulers enabling inter-grid usage. In this work they propose a Scheduling Description Language to extend the currently available job description language solutions. This work is still in progress, up to now only a partial SDL schema has been created. The meta-scheduling project in LA Grid [15] aims to support grid applications with resources located and managed in different domains. They define broker instances with a set of functional modules: connection management, resource management, job management and notification management. These modules implement the APIs and protocols used in LA Grid through web services. Each broker instance collects resource information from its neighbors and save the information in its resource repository or in-core memory. The resource information is distributed in the Grid and each instance will have a

view of all resources. The Koala grid scheduler [16] was designed to work on DAS-2 interacting with Globus [2] middleware services with the main features of data and processor co-allocation; lately it is being extended to support DAS-3 and Grid'5000. To inter-connect different grids, they have also decided to use inter-broker communication. Their policy is to use a remote grid only if the local one is saturated. In an ongoing experiment they use a so-called delegated matchmaking (DMM), where Koala instances delegate resource information in a peer-2-peer manner. Gridway introduces a Scheduling Architectures Taxonomy [17], where they describe a Multiple Grid Infrastructure. It consists of different categories, we are interested in the Multiple Meta-Scheduler Layers, where Gridway instances can communicate and interact through grid gateways. These instances can access resources belonging to different administrative domains (grids/VOs). They pass user requests to another domain when the current is overloaded – this approach follows the same idea as the previously introduced DMM. Gridway is also based on Globus, and they are experimenting with GT4 and gLite [3]. Comparing the previous approaches, we can see that all of them use a new method to expand current grid resource management boundaries. Meta-brokering appears in a sense that different domains are being examined as a whole, but they rather delegate resource information among domains, broker instances or gateways. Usually the local domain has preference, and when a job is passed to somewhere else, the result should be passed back to the initial point. Regarding multi-grid usage, the existing grids are very strict and conservative in the sense that they are very reluctant to introduce any modification that is coming from research or from other grid initiatives. Hence the solutions aiming at inter-connecting the existing brokers through common interfaces require a long standardization procedure before it will be accepted and adapted by the various grid communities. On the other hand the advantage of our proposed meta-brokering concept is that it does not require any modification of the existing grid schedulers, since it utilizes and delegates broker information by reaching them through their current interfaces. The HPC-Europa Project researchers also considered taking steps towards meta-brokering [18]; currently we have an ongoing work together with them to define a common meta-brokering model.

6 Conclusions

The Grid Meta-Broker itself is a standalone Web-Service that can serve both users and grid portals. The presented enhanced, adaptive scheduling solution with this Meta-Broker enables a higher level, interoperable brokering by utilizing existing resource brokers of different grid middleware. It gathers and utilizes meta-data about existing widely used brokers from various grid systems to establish an adaptive meta-brokering service. We have developed a new scheduling component for this Meta-Broker called Decision Maker that uses weighted functions with random generation to select a good performing broker to user jobs even under high uncertainty. We have evaluated the presented algorithms in a simulation environment based on GridSim with real workload samples. The presented evaluation results

affirm our expected utilization gains: the enhanced scheduling provided by the Decision Maker enables better adaptation and results in a more efficient job execution.

7 Bibliography

References

- [1] Foster, I. and Kesselman, C. *Computational Grids, The Grid: Blueprint for a New Computing Infrastructure*, pp. 15–52, Morgan Kaufmann, 1998.
- [2] Foster, I. and Kesselman, C. *The Globus project: A status report*, pp. 4–18, In Proc. of the Heterogeneous Computing Workshop, IEEE Computer Society Press, 1998.
- [3] *EGEE middleware technical website*, <http://egee-technical.web.cern.ch/egee-technical>
- [4] Erwin, D. W. and Snelling, D. F. *UNICORE: A Grid Computing Environment*, pp. 825–834, In Lecture Notes in Computer Science, volume 2150, Springer, 2001.
- [5] *Job Submission Description Language (JSDL)*, <http://www.ggf.org/documents/GFD.56.pdf>
- [6] Kertész, A., Kacsuk, P., Rodero, I., Guim, F. and Corbalan, J. *Meta-Brokering requirements and research directions in state-of-the-art Grid Resource Management*, Technical report, TR-0116, Institute on Resource Management and Scheduling, CoreGRID – Network of Excellence, November 2007.
- [7] Kertész, A. and Kacsuk, P. *Grid Meta-Broker Architecture: Towards an Interoperable Grid Resource Brokering Service*, pp. 112–116, CoreGRID Workshop on Grid Middleware in conjunction with Euro-Par 2006, Dresden, Germany, LNCS, Vol. 4375, 2007.
- [8] Kertész, A. and Kacsuk, P. *A Taxonomy of Grid Resource Brokers*, pp. 201–210, 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS 2006), Springer US, 2007.
- [9] *Open Grid Forum (OGF) website*, <http://www.ogf.org/>
- [10] *OGF Grid Scheduling Architecture Research Group*, <https://forge.gridforum.org/sf/projects/gsa-rg>
- [11] Knuth, Donald E. *The Art of Computer Programming Volume 2.*, Section 3.2.1. Addison-Wesley Professional, 1997.
- [12] Buyya, B. and Murshed, M. *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*, pp. 1175–1220, Concurrency and Computation: Practice and Experience, Volume 14, Issue 13-15, 2002.
- [13] Howell, F. and McNab, R. *SimJava: A discrete event simulation library for Java*, In Proc. of the International Conference on Web-Based Modeling and Simulation, San Diego, USA, 1998.
- [14] *Parallel Workloads Archive website*, <http://www.cs.huji.ac.il/labs/parallel/workload/>
- [15] Rodero, I., Guim, F. and Corbalan, J., Fong, L.L., Liu, Y.G. and Sadjadi, S.M. *Looking for an Evolution of Grid Scheduling: Meta-brokering*, Coregrid Workshop in Grid Middleware'07, Dresden, Germany, June 2007.
- [16] Iosup, A., Epema, D.H.J., Tannenbaum, T., Farrellee, M. and Livny, M. *Inter-Operating Grids through Delegated MatchMaking*, In proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC07), Reno, Nevada, November 2007.
- [17] Vazquez, T., Huedo, E., Montero, R. S. and Llorente, I. M. *Evaluation of a Utility Computing Model Based on the Federation of Grid Infrastructures*, pp. 372–381, Euro-Par 2007, August 28, 2007.
- [18] *The HPC-Europa Project website*, <http://www.hpc-europa.org>

Received September 10, 2008