Case-Study for Different Models of Resource Brokering in Grid Systems

Salman Toor[1], Bjarte Mohn[2], David Cameron[3], Sverker Holmgren[1]

[1] Div. of Scientific Computing, Dept. of Information Technology, Uppsala University Sweden,
[2] Div. of Nuclear and Particle Physics, Dept. of Physics and Astronomy, Uppsala University Sweden,
[3] Nordic Data Grid Facility (NDGF), Denmark
email: {Salman.Toor, Sverker.Holmgren}@it.uu.se Bjarte.Mohn@fysast.uu.se David.Cameron@cern.ch

Abstract

To identify the best suitable resource for a given task in a grid system, under constraints of limited available information, requires an elaborate strategy. The task gets even more complicated if the environment is heterogeneous and the availability of the resources or its information is not guaranteed for all the time. Efficient and reliable brokering in grid systems has been discussed extensively, and different strategies and models have been presented. However, this issue still needs more attention. In this paper we first review different brokering models, compare them and discuss the issues related to them. We have identify two key areas for improving the resource allocation: The **Information Flow** in the system and the **Data Awareness** of the resource selection. Our results show that the better management information flow between different components in the grid system significantly improves the efficiency of the resource allocation process.

1 Introduction

Grid systems have emerged as a useful environment to address requirements of large scale computing. The idea of utilizing geographically distributed resources is itself very interesting and contains many challenges which have been addressed over the last couple of years. A study of grid systems bring up a number of challenges, for example providing system information and monitoring, managing local sites running under different policies, providing system security and simple and easy-to-use clients, fault tolerance and efficient and reliable resource allocation according to the job requirements. In a grid system, the computational resources can be anything ranging from single computer to a very high-end cluster and storage resources can consist of a single disk or a complex RAID storage. There are number of production grids available which allows a seamless view over distributed resources and handles very large amount of jobs from different scientific real and virtual experiments in e.g. particle physics, quantum chemistry, bioinformatics etc.

One of the important areas in computational grids is resource allocation, and it is widely accepted that efficient and reliable resource allocation is necessary for the success of any grid middlewares. A number of brokering models has been studied and implemented in different grid system.

The task for resources brokering is to hide the complexity of the underling system and select the best possible resource from the available pool. This require lots of input and output from the other components in the grid system, such as the information and cataloging components, and sometimes also directly from the resources (depends on the architecture of the system). In many cases it has been observed that the brokering component is a scalability bottleneck or single point of failure in the whole grid system. A very tight connection between different components in the grid system results in that the overall performance is affected, while a too loosely coupled approach instead can have an effect on the reliability of the system.

A study of the brokering models in different systems like gLite, Condor-G, ARC and Nimrod/G reveals that it is not the information-providing component itself, but the actual information flow within the system which has an effect on the overall performance. There is always a trade-off between maintaining more up-to-date information about the resources and the amount of communication overhead between the information system and individual resource. For this purpose different schemes for information and monitoring has been developed and are in used by different middlewares. One of the conventional approaches is to build a hierarchical information system in which different resources are registered with multiple information systems and then those information systems are further registered with higher level of information systems. This hierarchical approach allows information to be fetched at different levels.

In this case study our aim is to analyze the differences between brokering models used within different middlewares. This will highlight the strengths and weaknesses of the models. We also propose some key modifications in the brokering component of the ARC middleware, and our results show that these modifications improve the efficiency the brokering component which in turn directly has an impact on the overall user response time. In the rest of the article, first we give an overview of popular brokering models, and then we cover the issues related with these models. Then we briefly explain the architecture of the ARC middleware, its components connected with the resource brokering and the proposed modifications. Section 6 covers results and discussion and section 7 describes our future direction of work.

2 Different Brokering Models

The question of selecting the best resource for a given set of tasks has been addressed using many different approaches even before the term computational/storage grids was invented. The process has been studied in various distributed systems at different levels [7, 12, 25]. In [22], each component of the grid system is characterized on the basis of its role and attributes. For example, the

resource discovery can be characterized as an agent-based systems or a query-based systems, and a further classification into centralized and distributed querying systems is made. The resource scheduling taxonomy contains four categories (Scheduler Organization, State Estimation, Rescheduling, Scheduling Polices) which are further subdivided according to their roles and attributes. Scheduler Organization can be Centralized, Hierarchical and Decentralized; State Estimation can be Predictive or Non-predictive, which can further be classified into Heuristics, Price Model, Machine Learning etc.

Apart from the abstraction resulting from defining theoretical taxonomies, a number of grid middlewares are currently using different models for resource allocation [8] in production environments. It is important to note that the resource allocation and the actual task submitted to selected resource are two separate processes. The grid resource broker selects a resource on the basis of the available information and is also known as the high-level or meta-schedule, whereas the Local Resource Management System (LRMS), or the low-level scheduler, is responsible for submitting jobs to the underlying cluster. For the mete-level scheduler, centralized or distributed brokering can be used. On top of these models, several more elaborate schemes and hierarchical models have been developed. The hierarchical approach is a combination of the centralized and distributed model. Here, the responsibilities are distributed across a hierarchy of schedulers. Schedulers belonging to the higher levels of the hierarchy make scheduling decisions concerning larger sets of resources (e.g., the resources in a given continent), while lower-level schedulers are responsible for a smaller set. Finally, at the bottom level of the hierarchy are the schedulers that schedule individual resources.

Over the time middlewares have become more and more stable and a large number of projects have ported their application to grids. These applications have different requirements. For example, a mission critical application requires the results in certain time frame and the workflows of complex application uses workflow models which also require scheduling on the application level. Such requirements add another level of complexity, and as one means to handle such situations the concept of advanced reservations [14] has emerged.

Using an economy driven approach adds another possibility of scheduling computational resources. The implementation of this model can be done by approaches mentioned above. The concept is to create a virtual market in which time for utilizing resources can be purchased from the resource providers, where the prices varies according to the resource demand, as it happens in the real market. A number of different projects have been initiated to study the different aspects of the computational economy [6, 23, 20]. In [6] Nimrod/G's resource broker is described. This has a hierarchical model with market based strategy, while as Tycoon [23] exploits the same idea with an agent based model.

In the following sections the discussion will be focused on the technical implementation of the resource broker modules together with the role of the components required to find the potential resource within different production level and research based grid systems.

2.1 Centralized Brokering

In the centralized brokering model, one entity is solely responsible for allocation of all the available resources. The generic architecture of the central resource broker is presented in figure 1. The components of the broker varies depending on the features provided by the system. The following example will show the strength of this approach:

glite [1] is the production level middleware developed by the EGEE projects and used by the European Organization of Nuclear Research (CERN) and their collaborative partners. gLite consists of a set of different components based on a Service Oriented Architecture (SOA) that all together provides the functionality [26]. The Workload Management System (WMS) [29] in gLite is responsible for distribution and management of the tasks across available resources. The features provided by the WMS requires some more components which are not covered in generalized model shown in the figure 1. For the details please refer to figure-1 in [29].

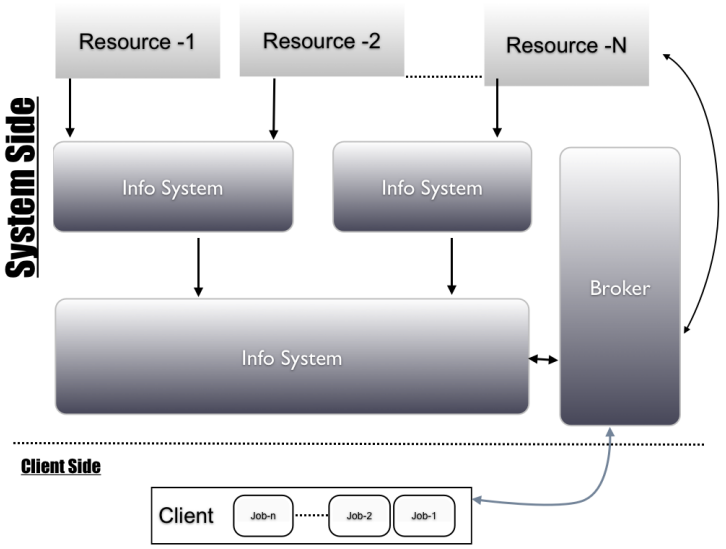

Fig. 1: Centralized Brokering Model

When a client submits a job in gLite it basically passes the submission responsibility to the WMS. WMS then selects an appropriate resource and send the jobs to that resource. WMS also keeps a persistent view of all the jobs submitted to it and take decisions if an unexpected situation occur during the job runtime. In the case when the matchmaking component cannot find a suitable resource the job is kept in the queue and submission is retried later. If the job fails, a resubmission process is automatically started by the WMS. Apart from

reliable job submission, WMS also allows user requests such as job cancelation and retrieving job output. The WMS is tightly connected with the Logging and Book Keeping service [15]. To compensate for the delays in the information system and not to overload the computing resources, the matchmaking component uses a "fuzzy rank" stochastic algorithm to perform a kindo fo random resource selection from the available candidate resource list. Using a "gang-match" function the matchmaking component also allows other characteristics, for example location of the input data, to be consider while selecting the resources. The centralized control of the WMS allows for both a pull and push mode for job submission. The Information Supermarket (ISM) component in WMS gives the freedom to implement different polices on resources. This component maintains a pool of available resources with some set of policies. In WMS, first both jobs and resources are structured in the queue and then with the help of lazy scheduling the jobs are assigned to the resources. This approach allows reliable handling of single and bulk jobs as well as complex workflows. Another middleware that uses a centralized brokering model is Condor [27].

2.2 Distributed Brokering

The distributed brokering model is also sometimes referred to as user-centric brokering. In this approach each user uses its own broker for finding the best candidate for the jobs. A broker is the a stateless, portable entity tightly connected with the client, and this shift the resource brokering functionality towards the client side rather than to server side. Figure 2 shows a generalized model of a distributed resource broker.

The Advanced Resource Connector (ARC) [13] is a project initiated within the NorduGrid collaboration for fulfilling the needs of large scale computing by connecting the geographically distributed resources and build a computational grid. The whole middleware consist of three main components: The Grid Manager works at the computing resource side, receive jobs and dispatch them to the Local Resource Management System (LRMS). The Information System is a hierarchical information store which is used to store the URLs of the available resources (for computing and storage), and the Client Side Brokering component finds the best candidate resource from the available resource list.

When a client requests to submit the job in ARC it first contacts the information service to get a list of the available resources. This list contains the URL(s) of the available resources. Once the client gets the list it contacts each individual resource and get the all the information related to the resource from the Local Information Service (LIS). This information is then passed to the Broker components at the client side. On the basis of the retrieved information the broker rank the resources and then tries to submit the jobs to the highest ranked resource. If the jobs cannot be submitted to this candidate resource, then the next possible resource will be tried until the job gets submitted.

The ARC example explains how the true distributed resource brokering model works. This approach give enables the use of a very thin and portable brokering component. The model also has the potential of scaling well since

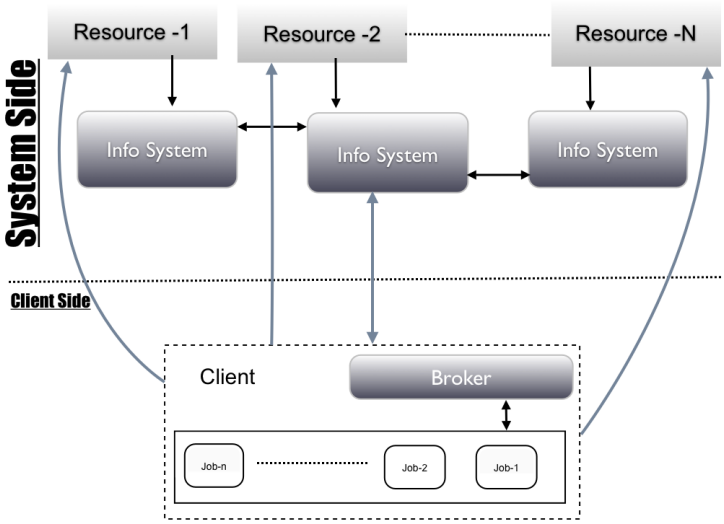

Fig. 2: Distributed Brokering Model

it divides the communication overhead over the clients which are distributed. Since in this model each broker is responsible for one or a set of jobs from an individual user there is no need to have a queuing system for the jobs.

2.3 Hierarchical Brokering

The hierarchical broker model combines the centralized and distributed approaches. The idea of a hierarchical model is not to have one big single resource broker, providing lots of functionality, and neither a very lightweight component with minimal functionality. Instead, the idea is to have multiple instances of a broker component distributed on the basis of some set of rules. Figure 3 depicts the generalized structure of the hierarchical model.

There are number of projects in which this approach has been used for resource allocation. Several different approaches are adopted for creating these hierarchical structure. For example, a virtual organization- based [16] hierarchical structure is presented in [21] and a different middleware-based hierarchical structure in [19].

The concept presented in [21] exploits a VO hierarchy in which VO(s) are composed of sub-VO(s) etc, and at the leaf-level in this hierarchy, groups of resources which are considered as a uniform system are placed. This hierarchical structure defines the hierarchy for the resource brokers. When the request comes to a high-level broker corresponding to a high-level VO in the tree, it will be passed further down to the broker connected with the right sub-VO in the hierarchy.

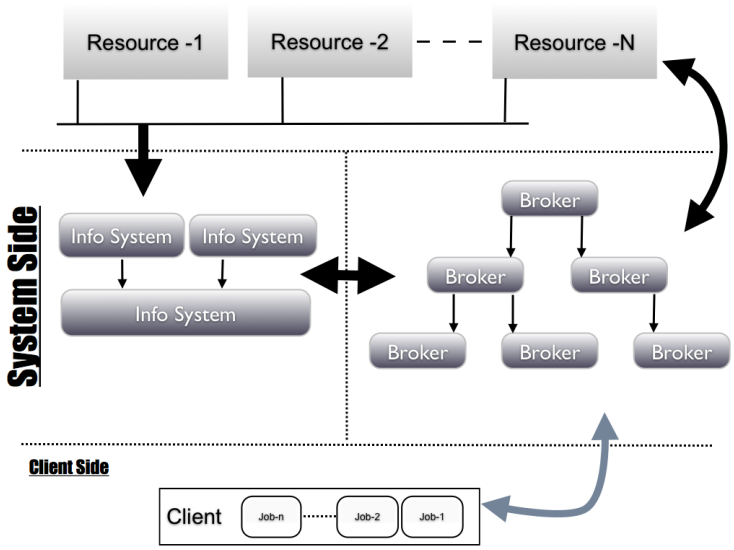

Fig. 3: Hierarchical Brokering Model

In [19] another hierarchical structure is presented for the use of different middlewares by the same set of client interfaces. The project based on the tools provided by the Globus Tookit. The GridWay [2] meta-scheduler is used for the resource brokering. GridWay enables submission of jobs to different LRMS as well as middlewares such as AstorGrid and gLite. In [19] a new scheduler-adaptor has been developed which allows Globus GRAM (Grid Resource Management service which is the core pillar of the Globus Toolkit) to send jobs to GridWay which then uses its brokering algorithm to submit jobs to LRMS or on different middlewares.

Using a hierarchical approach the congestion resulting from the handling of all requests at the same level (the approach followed in the centralized model) will be avoided and at the same time the control available at each hierarchical level makes it possible to provide some features that are available in the centralized approach.

2.4 Agent Based Brokering

The agent based resource brokering model has been examined a number of times in the Grid world [10, 11, 17, 33, 32]. Agents are software components considered to have intelligence, autonomous in nature, capability of self-healing and taking decisions. The illustration in Figure 4 describes a generic architecture for agent-based models in grid systems. The dynamic nature of agents makes this approach different from the conventional distributed and hierarchical model discussed above. The interaction between the agents within the system can be

of peer-to-peer, hierarchical or master-slave based.

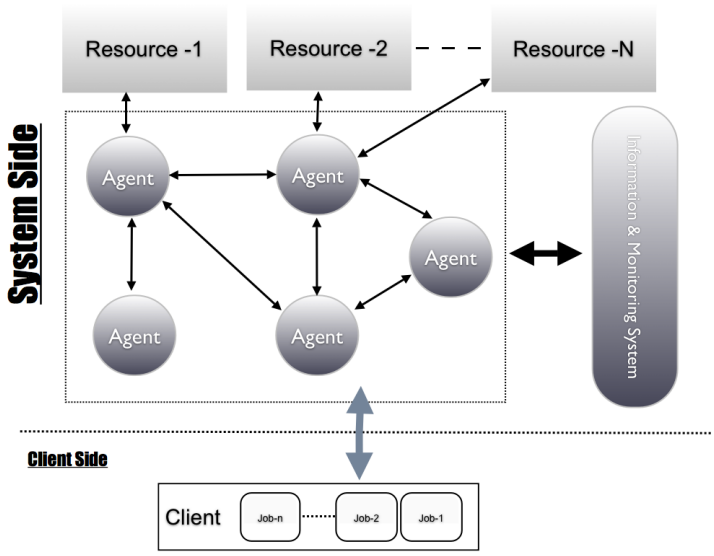

Fig. 4: Agent-based Resource Brokering Model

The approach presented in [10] uses a hierarchical structure of agents that together provides efficient resource management in the system. In the Agent Resource Management System (ARMS) each agent is responsible for a single or a set of resources, which means that the agents are basically service providers. Information about resources are exposed by the agents and the communication between them provides a mechanism for resource discovery. According to the model described in [10], each agent maintains an Agent Capability Table (ACT) for the resources connected to it as well as the other registered agents. The communication amongst the agents can be push- or pull-based. When a request comes to an agent it uses a matchmaking algorithm to select the best resource connected with the agent. If an agent cannot find a suitable resource, the request will be passed on to the agent higher in the hierarchy. The load on the agents is monitored by an option agent called Performance Monitoring Agent (PMA). This special agent analyzes the load using a built-in modeling and simulation mechanism and adoptively selects new policies for better load balancing.

Another agent-based approach for efficient resource brokering is discussed in [33, 32]. Here, a master-slave model is used to create teams which offers same type of services. The team leader (known as LMaster) subscribe its team to the central information system called CIC (Client Information Center). In [32] three strategies are discussed to get the best benefit out of this scheme for efficient resource brokering in grid systems.

The examples above show that within an agent-based model, some structure

corresponding to a classical model is used (centralized, Distributed and Hierarchical) and combined with the decision making capability of agents a scheme for addressing the problem of efficient and reliable resource allocation and management in the heterogeneous environment of computational grid is set up.

3 Brief Comparison Between Models

Each model reviewed above have pros and cons. The advanced functionality possible using a centralized model comes with the cost of scalability and performance issues. The limited scope of a distributed broker restricts the use of many fancy brokering features. An hierarchical approach can be effective, but the division of tasks and responsibilities need to be handled carefully. It has been also observed that the amount of time required to identify the candidate resources using hierarchical approach is larger than for the other approaches. Having knowledge-based or system-aware, autonomous agents in the system can ideally make it possible to manage the resource allocation in an efficient and reliable way. However, the knowledge of those agents comes with the cost of heavy communication amongst them.

The discussion above on different brokering models reveals that the problem of resource allocation is not only related to the selection mechanism but perhaps more dependent on the information available to the broker and how up-to-date this is. Maintaining updated knowledge of the status of the resources can be quite expensive and the cost increases as the size of the system increases. The communication created by the information flow in the system does not only have an effect on the process of resource allocation but also on the overall efficiency.

4 Proposed Model

In our study above we have identified distributed and hierarchical models as a potential candidate for building a robust and reliable resource allocation component. Such a model has been implemented in the next generation of the Advanced Resource Connector (ARC) middleware. Below we present a modification of the original distributed resource allocation model which significantly improves the efficiency of the resource brokering component. We will first give a brief overview of the next generation ARC middleware and the components involved in the process of resource selection, and then we will present the new model for the resource allocation.

4.1 Advanced Resource Connector (ARC)

The next generation of the Advanced Resource Connector (ARC) Grid middleware is developed by the NorduGrid [4] consortium and the EU supported KnowARC project [3]. It is built on a Service Oriented Architecture (SOA) in which services run in a customized service container called the Hosting Environment Daemon (HED) [9]. HED comprises pluggable components which provides

different set of functionalities. The Data Management Components are used to transfer the data using diffeernt protocols, the Message Chain Components are responsible for the communication within clients and services, the ARC Client Components are plug-ins used by the clients to connect to different Grid flavors, and the Policy Decision Components are responsible for the security model within the system. There are number of services available to fulfill the fundamental requirements of the grid system. For example, Grid job execution and management is handled by the A-REX service, policy decisions are taken by the Charon service, the ISIS service is responsible for information indexing, and batch job submission is handled by the Sched service. Further details about each of the component and services is presented in [5].

4.2 A-REX

The A-REX (ARC Computational Job Management Component) [24] is the core component of the ARC middleware service stack. This service runs on the computational resource. The main task is to accept the incoming execution request and dispatch them to the local resource management system (LRMS). A-REX is also responsible for getting the input data required by the grid jobs as well as to put the output data at the requested location. When the A-REX services is started on the resource, it registers itse access information (ServiceEPR, ServiceType) to the information system. This information is then used by the clients to directly contact the computing resource. In response to the client request, the A-REX service gives detailed resource information and an ongoing activity map to the client. This detailed resource information and activity document is based on the OASIS Web Services Resource Propoerties Secification [30].

4.3 Information Service (ISIS)

The ISIS (Information System Indexing Service) [18] is an information-providing service in the next generation ARC middleware. Every generic service in ARC registers itself at an available ISIS service in the system. It uses the GLUE-2 [31] information model for publishing Information. Each ISIS service has an XML-based database which is used to store information about the registered services. All the available ISIS services form a peer-to-peer network in which the information about the registered resources/services is propagated in order to create a coherent view of the system. ISIS uses a push-based model in which services register for a limited duration and when the limit expires the service again sends its information to the ISIS. The same push-based model is used between different ISIS services in which when an ISIS receives new information it passes that information to its neighbor ISIS services. Currently ISIS register each service with the following information, ServiceEPR (Endpoint Reference), ServiceType, ServiceID, GenTime (Generation Time) and Expiration (Validity period of service information).

4.4 ARC Client Tool

The next generation ARC middleware supports plug-in components which provide various functionality such as handling user proxy, resource discovery and information retrieval, matchmaking and brokering and job submission and management. The new ARC client provides support for three different job description languages JSDL, JDL and XRSL.

ARC follows the distributed or user centric resource brokering model. When a user submits a job using ARC, client tool first contact to the ISIS to get the list of all the available resources. In the next step client simultaneously contact all of the registered resources and get detailed information about the resources. For each of the received response it does the matchmaking with the job requirement and select a list of potential candidate resources for the job. ARC brokering component also allows to plug-in a customized broker which can further filter the resource according to the user specific demand. The customized brokers can be written in C++ and Python. The detailed technical information of each of the functionality provided by ARC client tool is explained in [28].

5 Prototype Solution

In the existing model used in the ARC middleware a broker at the client side is used for selecting the candidate resource and submit jobs to it. In the proposed model we adopt a three layer brokering model as shown in figure 5. Our initial tests using the ARC client shows that as much as 90% of the job submission time was spent on the resource discovery and only 10% portion was used for the matchmaking and actual submission. The goal of creating a hierarchical model is to subdivide the responsibilities and minimize the time spent in the resource discovery which in turn enhance the efficacy of the resource allocation. For this purpose we have proposed following changes.

- Our studies show that the information attributes provided by the computing resource can be divided into two categories: Static and Dynamic Attributes. This characterization is based on the nature of the resource information attributes for example; Operating System is the type of attribute which will rarely change similarly machine architecture and number of core thus are quite static by nature. In contrast, number of submitted jobs and available amount of virtual and resident memory are the examples of attributes which are dynamic in nature. The identified static attributes are shown in the Table 1.
  We have considered *ApplicationEnviroment(s)* (provides information about the available softwares on the computing resource) as an important attribute in the set of static information. Using this subdivision of attributes it has been observed that the set of static attributes is significantly smaller then the set of dynamic attributes. The proposed taxonomy also allows to handle static and dynamic attributes differently.
- Currently in ARC, the ISIS only provides the resource access information such as URL and resource type. In the proposed model the resources

Static Attribute	Type
OperationSystem	Simple
TotalPhysicalCPUs	Simple
Platform	Simple
ApplicationEnvironment (AE)	Complex
HealthState	Simple

Tab. 1: Selected static attributes with their types

running A-REX registered with the ISIS send a set of information to the ISIS which include access information together with the static information. This modification allows for the division of the load of information flow within the system.

- In the proposed model, the process of resource discovery and selection is divided into two parts, first it gets the list of registered resources from the ISIS which contains the static informations, filter the resources by using the of static information and then contact the pre-filtered resources for the dynamic information.
- In the current brokering module in ARC, users can also create customizable broker. Using these customized broker further filtering can be achieved. This functionality is also part of the proposed model and such brokers can be plug-in at the lower level in the broker hierarchy.

After implementing the proposed modification, Figure 5 shows the process of job submission. When a user submits a job, the ARC client first contacts the ISIS and gets the list of all the registered computing resources. This information list contains the resource's access information together with the static attributes. Static Broker does the first level matchmaking between the client request and the static attributes of the available resources, and short list the resources for further processing.

This first level pre-filtering improves the overall submission process, i.e. if a job requires certain *Application/Software* then after first pre-filtering only those resources will be contacted which supports that *Software*. To contact each individual resource is an expensive operation and minimizing these connection improves the resource selection process. Once the client gets the dynamic information from the resources, the dynamic broker component does the second level matchmaking and generates a list of resources which can fulfill the client request. Since everything is handled at the client side, similar to the current approach used in ARC the proposed model also allows users to create their own customized brokers which can further short list the candidate resources. For example further selection can be on the basis of *FastestQueue*.

The proposed model is as simple as the current ARC brokering model, yet significantly enhance the efficiency in the overall resource selection mechanism. The attributes for static information need to be chosen carefully as the time to refresh the information in the ISIS is longer and we only have to select those attributes which are not going to change frequently. It is also important to have

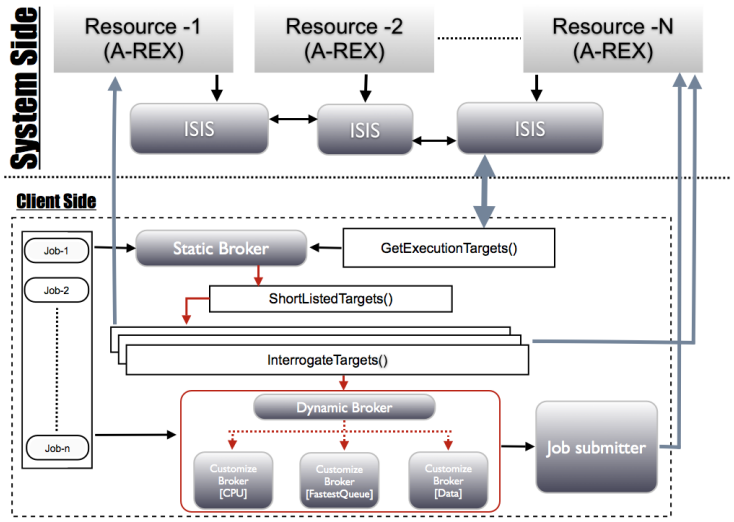

Fig. 5: Modification in ARC resource allocation

small list of attributes in static information list as it will create communication load while resources registering themselves to ISIS and even higher amount of load when client query the ISIS for registered resources.

6 Results and discussion

In order to evaluate the performance of the proposed model we have conducted a proof-of-concept test. Due to limited hardware resources we have used a small testbed with three computing resources and five grid jobs with different requirements. The test environment is small, but still sufficient to explain the concept and to measure the performance gained by the proposed model. The testbed consist of following machines:

Machine Name	Operating System	Application	Services	Location
M1	OSX 10.4.11	Matlab7.0	A-REX	Sweden
M2	S.L - 5.3	Matlab6.1	A-REX+ISIS	Sweden
M3	CentOS 5	APP-1	A-REX	Slovakia
M4	S.L - 5.3	-	Client	Sweden

Tab. 2: Resource description used in the test

The modified ARC client is installed on M4. M1, M2 and M4 machines are on a same network of Uppsala university Sweden whereas M3 is located in

Slovakia. This makes the communication between M1, M2 and M4 faster than communication with the M3 machine. It turns out that if client contact all the resources for the detailed information, M3 takes almost 85% of the overall information retrieval time. The combination of these machine will also help to demonstrate the handling of the resources which are on the slow network or having slow response time. Table 3 shows the requirements of five grid jobs.

Job	Requirements
1	No special requirements
2	OperatingSystem = Mac OSX
3	ApplicationEnvironment = Matlab6.1
4	ApplicationEnvironment = APP-1 and OperationSystem = Linux
5	ApplicationEnvironment = MatLab7.1

Tab. 3: Jobs with special requirements.

- Job-1: Simple test job, runs (/bin/true). Any resource can be the potential candidate for this job. For Job-1 the performance of old client and the new client should be (almost) the same. Using modified client after getting information from ISIS all the resources will be contacted for the dynamic information.
- Job-2: Requires MAC-OSX operating system to run the job. Only one machine will be queried for the dynamic information as operating system is in the static attribute list and fist level filtering will only allow to contact the M1 machine. Using the proposed modification, job-2 will save two extra connections (to M2 and M3) compared to the old client.
- Job-3: Requires Matlab6.1. The requested software is only available on M2 machine. Again only one machine will be contacted for the detailed information.
- Job-4: *ApplicationEnvironment*=APP-1 and *OperatingSystem*=linux. Both of these requirements will be fulfilled by M3 machine. The requirements of this job explicitly selects a machine which is comparatively slower than the other computing resources available. But still we are avoiding two extra connections to M1 and M2 for Dynamic information.
- Jo-5: Requires Matlab7.0, only available on M1 so after contacting to the ISIS only one detailed query will be required.

Avoid extra connections for the dynamic information is the key benefit of this model. In our test environment we have three computing resources, first query to ISIS is compulsory(required by both current and modified client) but after that by exploiting the small subset of static information on one resource is contacted for job 2,3,4 and 5 out of three. Theoretically this gives 66.667% better efficiency if the cost of connecting each machine is equal but this is not the case in grid infrastructure. But still by avoiding the extra connections we have gained better submission time as shown in the submission process of job

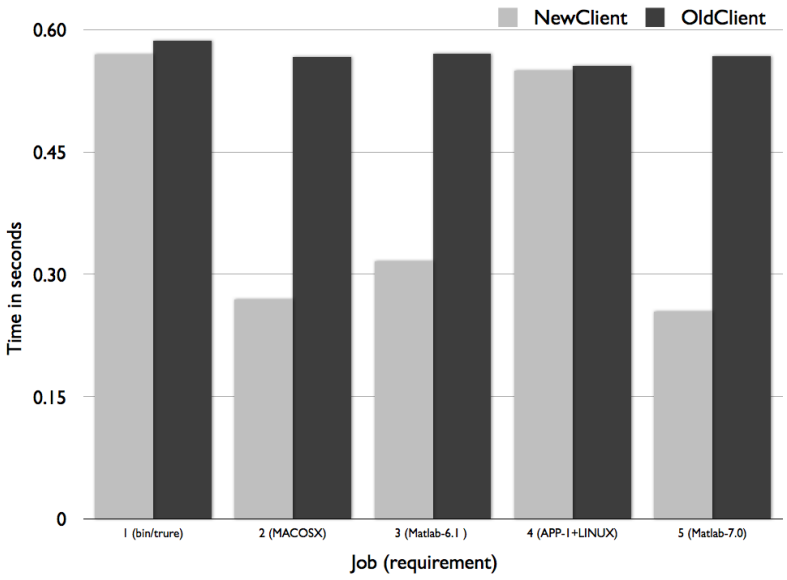

Fig. 6: System response time for selecting computing resource for different grid jobs having different job requirements.

2,3 and 5 in figure 6.

Listing 1: Attributes of Static Information

```
....
....
<StaticInfo>
    <OperatingSystem>Linux</OperatingSystem>
    <HealthState>OK</HealthState>
    <Architecture>i386</Architecture>
    <CPUMultiplicity>singlecpu-multicore</CPUMultiplicity>
    <PhysicalCPU>2</PhysicalCPU>
    <CPUModel>Intel(R) Pentium(R) 4 CPU 3.00GHz</CPUModel>
    <ApplicationEnvironments>
        <ApplicationEnvironment>
            <AppName>Matlab-1.6</AppName>
            <AppVersion>1.6</AppVersion>
            <AppID>urn:ogf:ApplicationEnvironment:uppmax.uu.se
            :sal6:Matlab-6.1/matlab</AppID>
        </ApplicationEnvironment>
    </ApplicationEnvironments>}
</StaticInfo>
....
....
```

Another important aspect to evaluate in the proposed model is the effect of response time for querying and registering service's to ISIS. As now there will be more attributes attached with each resource. Listing 1 show the static information portion of the information send to the ISIS. Using the proposed model when a client requests to get the information about the registered resources it

gets a similar set of information about each resource. It is important to note here that the*ApplicationEnvironment* contains more information then than the other attributes and it will grow with the number of softwares installed on the computing resources. In the GLUE-2 schema there are more attributes related with the *ApplicationEnvironment* but again for the static information we have only used the minimal static attributes which are essential.

We have compared the service registration process using only compulsory attributes (ServiceType, ServiceEPR, ExpirationTime), compulsory attributes together with 1, 10, 50 and 100 installed Application Environments. Table 4 shows minimum, average and maximum time taken by the service registration. Using each type of information 100 registration request has been made. According to the results the registration process has not been much effected by increasing the number of attributes in the registration process.

	Static Information				
Time	Compulsory	1 AE	10 AE	50 AE	100 AE
Minimum	0.1248	0.1248	0.1238	0.1250	0.1292
Average	0.1269	0.1283	0.1254	0.1258	0.1301
Maximum	0.12991	0.1908	0.1299	0.1281	0.1415

Tab. 4: Minimum, Average and Maximum time to register 100 services with only compulsory attributes, compulsory with 1, 10, 50 and 100 AE installed on each computing resource.

In the proposed model, client first contact the ISIS to gets the information of all the registered resources. Each computing resource now register itself to ISIS with more information. It is also important to measure the effect on the query response time when there are high number of resources registered with the ISIS and each resource has comparatively higher number of attributes then the previous model. Figure 7 illustrate the query response time, while including the number of resources from 10 to 1000 together with the number of attributes associated with each resource information. The size of the attributes is increased by including more Application Environments. The Information-axis show the attributes used by each resource to register itself to ISIS. For this test we have used the LAN setup in which a single ISIS client is used to fill the ISIS database with required information. Figure 7 shows the time required to get the information from ISIS. The response time can even gets more worst as it also highly depends on the connectivity between the client and the ISIS.

With the less number of registered resources the difference in the response time is negligible. But the results shows the high latency as the number of resources together with the information increased. This increase in query response time is expected and if we combine this latency with the gain we get by less number of connection for the dynamic information to each individual resource, makes this negligible, can be seen by figure 6. With the proposed model we have stretched the limit further away so that it helps to allocate the

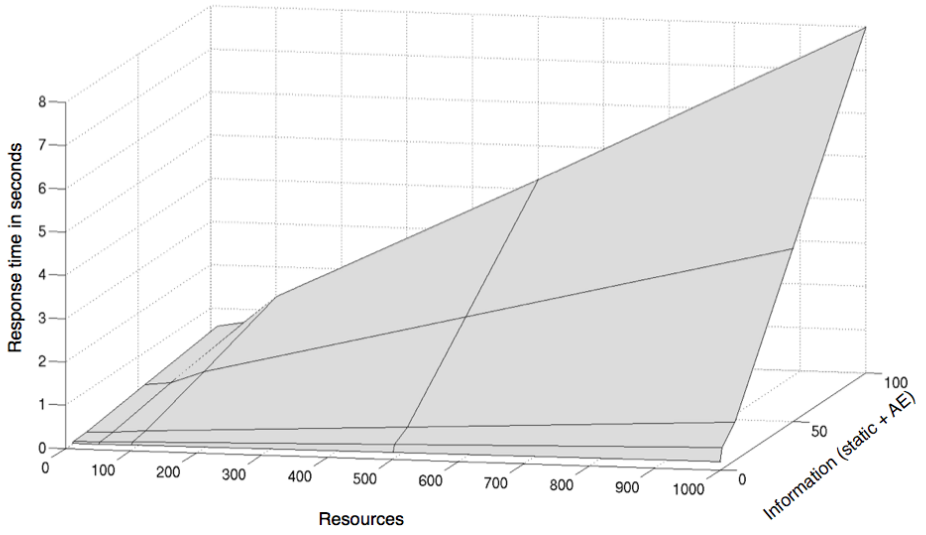

Fig. 7: The figure shows the ISIS response time while increasing the number of resources from 10 to 1000 on x-axis. Y-axis shows the static information used to register services to the ISIS. First we have registered all the services only with the compulsory attributes and then together with the details of 1, 10, 50 and 100 *ApplicationEnvironments*.

resources efficiently and reliably. Figure 6 depict the overall performance gain in the resource allocation process within a small test environment but the performance will be even more better if there are more resources in the system; i.e. if 100 computing resources are registered with ISIS and only 10 fulfill the static information criteria then there will be 90 less connections for the query of dynamic information. But at the same time if 90 fulfill the static requirements then the gain will be only 10% and in the extreme case the old and new client will end-up having same type of performance.

7 Future Directions

The result shows that the proposed model significantly improve the resource selection and overall job submission process. In the continuation of this work

we have plan to focus on following two directions.

First to test the proposed model for multiple job submission. The results presented in the case study are based on single job submission. It is often observed that the request to submit multiple jobs comes with almost same set of requirements(mostly with the same set of job parameters but with different application arguments). In such scenario the proposed model will also give better results. The model can even be more improved by creating a client side information cache for limited duration. This approach can help to reuse the information and avoid the unnecessary connections within a certain time.

In the second part the focus will be on the grid jobs which deals with the large data sets. The resource selection for the data intensive jobs requires certain strategy as to transfer the large data sets can be very expensive and time consuming. Thus in the second part the aim will be to study different available models for efficient handling of the data intensive grid jobs and made some improvements if possible.

References

1. glite: http://glite.web.cern.ch/glite/.
2. Gridway: http://www.gridway.org.
3. Knowarc project: http://www.knowarc.eu.
4. Nordu grid: http://www.nordugrid.org/.
5. Nordugrid papers: http://www.nordugrid.org/papers.html.
6. David Abramson, Rajkumar Buyya, and Jonathan Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource broker. *Future Gener. Comput. Syst.*, 18(8):1061–1074, 2002.
7. Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Blni, Albert I. Reuther, Mitchell D. Theys, Bin Yao, Richard F. Freund, Muthucumaru Maheswaran, James P. Robertson, and Debra Hensgen. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. *Heterogeneous Computing Workshop*, 0:15, 1999.
8. S. DiNucci D Buyya, R. Chapin. Architectural models for resource management in the grid. *LECTURE NOTES IN COMPUTER SCIENCE*, Volume 1971/2000:18–35, 1999.
9. D. Cameron, M. Ellert, J. Jönemo, A. Konstantinov, I. Marton, B. Mohn, J. K. Nilsen, M. Nordén, W. Qiang, G. Rőczei, F. Szalai, and A. Wäänänen. *The Hosting Environment of the Advanced Resource Connector middleware.* NorduGrid. NORDUGRID-TECH-19.
10. Junwei Cao, Stephen A. Jarvis, Subhash Saini, Darren J. Kerbyson, and Graham R. Nudd. Arms: An agent-based resource management system for grid computing. *Sci. Program.*, 10(2):135–148, 2002.
11. Junwei Cao, Daniel P. Spooner, Stephen A. Jarvis, and Graham R. Nudd. Grid load balancing using intelligent agents. *Future Gener. Comput. Syst.*, 21(1):135–149, 2005.
12. T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2):141–154, 1988.
13. M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced resource

connector middleware for lightweight computational grids. *Future Gener. Comput. Syst.*, 23(2):219–240, 2007.
14. Erik Elmroth and Johan Tordsson. Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. *Future Gener. Comput. Syst.*, 24(6):585–593, 2008.
15. A. Krenek L. Matyska M. Mulac J. Pospisil M. Ruda. Z. Salvet J. Sitera J. Skrabal M. Vocu F. Dvorak, D. Kouril. Services for tracking and archival of grid job information. In *Cracow Grid Workshop*, 2005.
16. Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.
17. Pascal Dugenie Stefano A. Cerri Frederic Duvert, Clement Jonquet. Agent-grid integration ontology. Volume 4277/2006(1):136–146, 2005.
18. Ivan Marton Gabor Roczei, Gabor Szigeti. *ARC peer-to-peer information system.* NorduGrid. NORDUGRID-TECH-21.
19. Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. A recursive architecture for hierarchical grid resource management. *Future Gener. Comput. Syst.*, 25(4):401–405, 2009.
20. Ying Li Feng Hong Ming Gao Jiadi Yu, Minglu Li. A framework for price-based resource allocation on the grid. Volume 3320/2005(1):341–344, 2005.
21. Donal K. Fellows John M. Brooke. An architecture for distributed grid brokering. Volume 3648/2005:475–484, 2005.
22. R. Buyya K. Krauter and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *SOFTWARE PRACTICE AND EXPERIENCE*, 32:135–164, 2002.
23. Bernardo A. Huberman Kevin Lai and Leslie Fine. Tycoon: A Distributed Market-based Resource Allocation System. Technical Report arXiv:cs.DC/0404013, HP Labs, Palo Alto, CA, USA, April 2004.
24. A. Konstantinov. *The ARC Computational Job Management Module - A-REX.* NorduGrid. NORDUGRID-TECH-14.
25. Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. Em3: A taxonomy of heterogeneous computing systems. *Computer*, 28(12):68–70, 1995.
26. E Laure, F Hemmer, F Prelz, S Beco, S Fisher, M Livny, L Guy, M Barroso, P Buncic, Peter Z Kunszt, A Di Meglio, A Aimar, A Edlund, D Groep, F Pacini, M Sgaravatto, and O Mulmo. Middleware for the next generation grid infrastructure. (EGEE-PUB-2004-002):4 p, 2004.
27. Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.
28. I. Marton G. Roczei M. Ellert, B. Mohn. *libarcclient A Client Library for ARC.* NorduGrid. NORDUGRID-TECH-20.
29. Cecchi Marco, Capannini Fabio, Dorigo Alvise, Ghiselli Antonia, Giacomini Francesco, Maraschini Alessandro, Marzolla Moreno, Monforte Salvatore, Pacini Fabrizio, Petronzio Luca, and Prelz Francesco. The glite workload management system. In *GPC '09: Proceedings of the 4th International Conference on Advances in Grid and Pervasive Computing*, pages 256–268, Berlin, Heidelberg, 2009. Springer-Verlag.
30. OASIS. *OASIS, OASIS Web Services ResourceProperties specication.*
31. Felix Ehm Laurence Field Gerson Galang Balazs Konya Maarten Litmaath Sergio Andreozzi, Stephen Burke and Paul Millar. *GLUE Specification v. 2.0, The*

Open Grid Forum, GFD-R-P.147, 2009.
32. Maria Ganzha Maciej Gawinecki Ivan Lirkov Svetozar Margenov Wojciech Kuranowski, Marcin Paprzycki. E?cient matchmaking in an agent-based grid resource brokering system.
33. Maria Ganzha Maciej Gawinecki Ivan Lirkov Svetozar Margenov Wojciech Kuranowski, Marcin Paprzycki. Agents as resource brokers in grids forming agent teams. *Future Gener. Comput. Syst.*, Volume 4818/2008(1):489–491, 2005.