An Architecture for Distributed Grid Brokering

John M. Brooke and Donal K. Fellows

University of Manchester, Manchester, UK
j.m.brooke@manchester.ac.uk
http://www.unigrids.org/

Abstract. Computational resource brokering on the Grid is the process of discovering what systems are capable of running a job, obtaining estimates for when that job may run and how much it will cost, and submitting the job to the system that best meets the users' requirements. This paper identifies how resource brokers differ from superschedulers, and describes a resource brokering architecture which is adapted to the emergent structure of the large-scale Grid. We outline the architecture of the UNICORE resource broker which is the basis of our prototype implementation, and discusses both how the existing UNICORE architecture is relevant to the wider brokering picture and what will be done in the future to bring them into closer alignment.

1 Introduction

If one examines currently accepted informal definitions of Grid computing (e.g. [1]) it is clear that the Grid approach to distributed computing does not allow us to make any assumptions about uniform policies for the management of and access to the resources provided by participation in the Grid. This potentially makes Grids very hard to use and a great deal of effort has been devoted to developing "middleware" that can hide this complexity from the users of the Grid. One important task of this middleware is to locate resources on the Grid for the purposes of performing some computational task. This is usually referred to as resource brokering and it usually also includes obtaining some form of quality-of-service (QoS) offer from those resources, so that different offers from providers may be distinguished.

This is a separate problem from the management and scheduling of tasks on Grid resources, which is referred to as "super-scheduling" since the Grid-wide scheduler may need to coordinate local scheduling systems given the potential existence of different management regimes controlling the resources of the Grid. Note that this condition makes Grid scheduling a different problem to scheduling on hierarchical clusters controlled by a uniform resource management system.

In some current resource broker implementations, e.g. in the EU DataGrid[2] and in NorduGrid[3], these two operations are merged so that the process of finding where a job can run is also the process of reserving some resources for that job. It is known that the super-scheduling problem is hard to scale especially when the local schedulers may be running different scheduling systems. The few

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 475–484, 2005.

super-schedulers that do exist (e.g. [4]) only work with certain batch or resource management systems, thus the problem is not generally solved in practice. Thus if a new site wishes to join in a larger Grid, it may have to consider changing its scheduling system and software, this breaks the autonomy of local site policy that is regarded as a key distinguishing feature of the Grid.

In this paper we argue that the resource broker problem is inherently easier to scale and present an architecture for brokering that follows natural hierarchies created when different sites or organisations which have a uniform inter-site policy join to form a Virtual Organisation (VO) by pooling their resources. We distinguish two types of scaling; scaling in magnitude allowing very large numbers of sites and resources to be organised as a Grid, and scaling in complexity allowing complex applications to federate across sites deploying different middleware and differing site policy configurations. We present preliminary results showing how our brokering architecture enables scaling in complexity and generalise from this to a proposed architecture that can support both types of scaling.

The paper is structured as follows: in Sect. 2 we examine in more detail current approaches to the scheduling and brokering problems. In Sect. 3 we present our VO-based approach to brokering and in Sect. 4 we present our reasons for choosing the UNICORE middleware as the basis for implementation. In Sect. 5 we present the detail of a prototype implementation and how this allows interoperability with other middleware systems such as Globus Toolkit versions 2 and 3 (GT2 and GT3). We also discuss how the lessons learnt have enabled a generalisation of the prototype implementation to meet the full requirements of a VO-based architecture. In Sect. 6 we present preliminary conclusions and discuss how the development of standards based around the Open Grid Services Architecture will enable the brokering architecture to be developed for a much wider range of middleware systems compliant with the emerging Open Grid Services Architecture (OGSA).

2 Existing Resource Brokers and Superschedulers

There are currently several different approaches to Grid resource brokering. Two major but similar approaches are used by the brokers developed in the DataGrid project and the NorduGrid project cited above, which are both arranged as front-ends to the job-submission system through which all jobs are required to go. In the DataGrid broker, there is a single central broker through which all submissions are made; this allows it to carry out not just brokering but also scheduling across the Grid which it controls, but it pays for this by being not scalable. The NorduGrid broker by contrast puts the brokering directly in the client toolkit; this solution is more scalable, but requires revealing large amounts of information to every client, which not something that commercial resource providers may wish to do (the precise status of a site might be commercially sensitive.) The problem with both of these approaches is that they both require collecting of large amounts of information about the state of the Grid in some centralized location, necessitating an extensive Grid monitoring architecture.

This approach does not scale well because of the amount of information that needs to be transferred and processed since a centralized broker must know the state of the system with a fair degree of accuracy or all estimates will be wrong, and a distributed broker needs to collect a large amount of information on each client request.

While both of the above two brokers are also scheduling agents, neither are as far advanced as the NaReGI[5] super-scheduler. That works through the introduction of a privileged module that can rewrite jobs into a form that can be scheduled more efficiently at individual resources. However, this approach also does not scale administratively because it requires the exposure of substantial amounts of information (much of which might be commercially sensitive) about resources on the Grid to the scheduler potentially within a different administrative domain. This approach can be extended with the use of an inter-scheduler negotiation protocol (e.g. based on ContractNet[6] or WS-Agreement[7]) but this is still difficult to scale as the system granularity changes.

The other approach in use is based on the UNICORE[8] resource brokering framework[9]). This assumes that the underlying systems used by the sites or organisations in the VO are each in control of local scheduling (e.g. through the use of a batch queue) and is oriented towards discovery of resources and the presentation of offers for a particular level of quality-of-service made by those resources. In particular, the resources may make multiple offers for a particular job and those offers do not have to precisely satisfy the requirements of the job in order to be considered; the resource making the offer may use knowledge of the application's problem-domain to create an offer based on application-specific scaling factors. The other key feature of the existing UNICORE broker is that the architecture is composable, with brokering agents being able to ask other brokering agents to work on their behalf.

3 Conceptual Basis for VO-Based Brokering

In [1] Grids are envisaged as deriving from resource sharing in Virtual Organizations. This term has no meaning unless the VO has some common policy on resource sharing, however this may be at a general level and scheduling, for example, may be a task carried out differently in different parts of the VO. If we define the VO concept recursively, i.e. VOs can be composed of sub-VOs, then we get a policy hierarchy. If we go sufficiently far down such VO trees we will eventually come to groupings of resource that can be considered to have uniform systems with respect to resource allocation and management (in the worst case this might be individual machines but economies of scale generally call for some grouping).

An actual physical computing site or organisation may have multiple sets of resources managed in substantially different ways, it can be easier to represent the site as multiple VOs, each with its own policy domain (though perhaps with a separate VO on top of them representing the federation of those resources within the overall site). Sites come together in organisations and organisations

in multi-institutional collaborations but in our abstraction this is all within the recursive definition of VOs. Note that resources are not necessarily machines, but are instead a virtualization of machines. This means that a resource may also be a cluster of machines, or that a single machine may host multiple resources (much as websites may be hosted by pools of webservers, or single HTTP daemons may host multiple websites; combinations of both are also possible).

Because we have defined the structure of VOs recursively, we also define the structure of the brokering system for the Grid recursively. By arranging for the broker for a VO to operate through delegation of requests sent to it to the brokers in the sub-VOs, there is already a much substantial degree of natural scalability. Another degree of scalability can be added by loosening up the binding between a VO and its broker, so that the brokering service for a VO is actually chosen from a pool of suitable brokers.

Superschedulers are integrated into this picture by placing them at (or near) the leaves of the VO tree. This allows them to operate in highly homogeneous environment and avoid the inter-domain coupling problems found in higher-level superschedulers. It is easier to scale brokers hierarchically across administrative domain boundaries, since they do not undertake any management of resources but only make enquiries about such availability. Thus the abstract function *getResourceInformation* can be implemented without any inter-site cooperation but *scheduleTask* cannot. This is not the case, however, in the hierarchical design of the Meta Directory System v2 (MDS-2) used by Globus middleware[10] since the indexing process requires that information publishing outside the subdomains of the VO and combined at the higher levels.

4 Choice of Middleware to Build a Hierarchical Broker

To build a Grid resource broker based on VO boundaries we need support for the hierarchical structure of VOs in the middleware which provides access to the resources of the Grid. We found support for such abstractions in the UNICORE middleware. The Globus MDS-2 information provision has a hierarchical structure but this abstraction is not maintained throughout the middleware (in job submission language for example). This means that when UNICORE is installed we simultaneously gain information about resources that covers all possible task submission requests (since the middleware cannot function without this). With Globus the information provision is done separately and although the adoption of a common information schema such as the GLUE schema[11] goes some way towards providing a more information-rich Grid, it still lacks the link between resource information gathering and task submission[12].

The UNICORE[8] architecture is based around the concepts of Usites, Vsites and Abstract Job Objects (AJOs). Usites are virtualizations of resource provider sites that will normally have a shared set of policies (originally focusing on firewall and certificate authority management), Vsites are virtualizations of services providing computational resources, and AJOs are document-oriented abstractions of computational jobs that are converted by Vsites into concrete forms (through a process termed "Incarnation") before execution (see Fig. 1). The

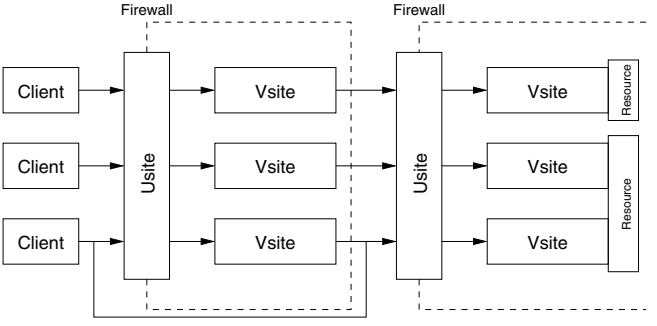

Fig. 1. The architecture of a UNICORE grid

UNICORE brokering model (developed in the EUROGRID project[9]) builds on top of the general architecture by allowing each Vsite to host a brokering service for that Vsite. This works by taking an AJO stripped of large components (like attached files) and testing to see if the resources it requests are available at the Vsite. When the resources are available, the broker then obtains an estimate for what level of quality-of-service is available for the job (obtained from the low-level job system, e.g. from a batch queue length estimator or by examining the load of the machine to get an estimate for likely processor and memory contention[1]) and then attaches it to a ticket that is handed back to the calling agent along with the QoS offer. The calling agent can then claim the offered QoS by attaching the ticket to the real job submission. Another key feature of the EUROGRID broker was support for delegation of a brokering request from one broker to another, which allowed for the deployment of a dummy Vsite which could provide brokering for a whole group of resources by delegating incoming requests to the leaf-Vsite brokers (see Fig. 2). Finally, it supports a plug-in interface which allows the broker to be enhanced with knowledge of a particular

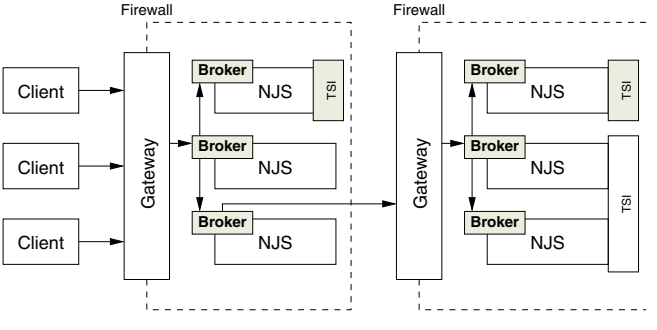

Fig. 2. The architecture of the EUROGRID broker

[1] Note that different kinds of systems require different kinds of QoS estimators. Batch processing systems give total control of processors to their jobs and hence the loading is irrelevant on such resources, whereas direct execution clusters will start running every job virtually immediately but will suffer from any resource contention present.

application domain. This allows for the expression of job requirements in terms of domain-specific measures (e.g. the size of Grid used in a weather simulation) and the application of performance models based on a detailed study of the actual applications in use, it being far easier to provide the user with an interface that generates such input metadata than it is to arrange for all agents on the Grid to make accurate physical resource requirement estimates.

The GRIP project[13] extended this model by allowing a Vsite to be implemented not just using the basic UNICORE mechanisms, but also on top of Globus[14]. This leveraged the fact that the AJO is abstract to allow the complete replacement of the job running system with another one with an entirely different job description language. The resource broker was also extended in GRIP to work by using Globus information services (in GT2 and GT3).

The key features of the UNICORE architecture that supported the EUROGRID/GRIP broker were that the conceptual models of both the computational resources and the jobs running on them were abstract. By brokering jobs before they are incarnated, it is much easier to find more resources capable of running the job on a heterogeneous Grid, and the abstract resource model allows the broker to work with offers from a much wider range of resources.

5 Design and Implementation

5.1 Prototype Implementation

The EUROGRID/GRIP resource broker is implemented as a plug-in module to the UNICORE NJS[2] and consists internally of a multi-layered architecture (see Fig. 3). The outermost layer handles the communication, security and delegation model as well as providing utility and configuration services. Inside this is is the main logic module — which uses a runtime-pluggable architecture to support application-specific brokering — and the local basic brokering engine, which acts as an interface to the underlying concrete system. The GRIP broker extends the EUROGRID broker in having additional local basic brokering implementations,

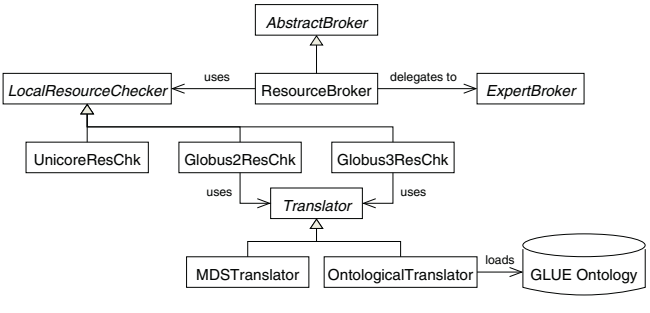

Fig. 3. The internal architecture of the EUROGRID/GRIP broker

[2] Network Job Supervisor, a hosting environment for UNICORE Vsites.

one that integrates with a GT2-based low-level Grid, and one that integrates with a GT3-based Grid. These local broker interfaces handle the task of brokering on a Globus-based Grid by translating from resources as requested by a UNICORE AJO to an MDS-2 or Index Service query. Each translation is performed by a pluggable translation engine; the translation to MDS-2 queries is *ad-hoc* but the translation to an Index Service query is described using an ontology developed using PCPACK[15], an ontology capture tool, which allows for much simpler maintenance of the ontology going forward in time.

To demonstrate the viability of the EUROGRID/GRIP resource broker, it has been used to broker the Deutcher Wetterdienst coupled weather simulation model across a heterogeneous grid consisting of a mixture of UNICORE-based and Globus-based nodes. The global part of the weather model was hosted on a UNICORE-based grid system, and the relocatable local weather model was transparently brokered across a Globus-based grid, with the results being reflected back to a UNICORE-based front end client for display to the user. This demonstrated both complexity of application (the primary resources over which the application was brokered were described in terms of the weather model, with translation to suitable underlying resource terms done transparently by the application-specific broker) and complexity of underlying infrastructure.

5.2 Lessons Learnt and Generalisation of the Architecture

Work on the prototype has identified three requirements to lift this infrastructure to the ideal of VO-based brokering architecture described in Sect. 3:

- The resource broker must be extended so that it can broker for more than one Vsite simultaneously without having to delegate to individual leaf brokers. This significantly reduces the degree of fan-out in a brokering request. This means that the tickets issued by the broker must be capable of inspection by services other than the issuer, though there is no need for anything outside the site that hosts such a broker to be able to carry out such an inspection.
- It must be possible to place two or more brokers in parallel and get sensible answers out of each even with simultaneous requests. This means that where a broker reserves resources for the use of a job, it must be able to make sure that the other brokers in parallel do not collide with it. This might be done by using a database to provide serialization and locking. Note that this is not necessary if no actual reservation is made for the job, such as might be the case if the brokers are just reporting estimates of how long it will take for the job to reach the head of the batch queue.
- It must be possible for an agent (whether a client or another broker) to find out an instance of a resource broker for a site or other VO. This should be done by associating a registry of some kind with the VO and placing the references to all the VO's broker instances within it. VOs that have more than one broker may wish to split the load between the brokers by arranging for different requests to the registry to return different instances (or in a different order).

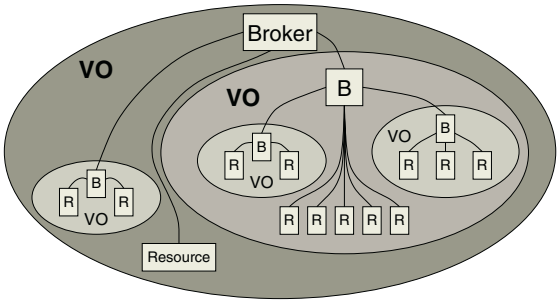

Fig. 4. An overview of the multi-VO resource broker architecture

These three architectural changes, together with the delegation model developed in the EUROGRID broker (formalized by an Explicit Trust Delegation model[16]), allow the development of a brokering infrastructure along VO lines (see Fig. 4). This work is being undertaken in the UniGrids project[17]. Note that the VO closest to the user should also be supplied with a policy description (based on Condor ClassAds[18]) that allows it to choose between the offers collected on behalf of the user. By combining that policy with any such VO-based policy, it is possible to choose a suitable offer without any further intervention from the user or instigating agent.

There are additional benefits to doing this. By moving the broker the higher level, it becomes much more efficient to use systems like R-GMA[19] or NWS[20] to estimate likely performance. The Usite level (i.e. the lowest level of VO) is a good point to introduce superschedulers like that developed by NaReGI[5] without sacrificing the simplicity of the wider brokering architecture outlined above.

The final component of the UniGrids brokering architecture is a mechanism for monitoring of submitted jobs and generation of Usage Records[21]. These usage records would then be both stored in a resource usage service (such as the one developed in the MCS project[22]) for future reference (e.g. invoicing the VO at the end of the month). The usage records are also to be fed back at periodic intervals at times when the brokers are likely to be otherwise lightly loaded.

Preliminary results indicate that the new architecture greatly increases both the throughput scalability and the management scalability of the brokering system. The key source of improvement in throughput is the reduction in the number of inter-service messages achieved through a broker being able to issue offers on behalf of an entire site at once, and the improvement in managability comes through the reduction of the number of different systems that need to be configured to bring a brokered site into service and keep it operating.

6 Conclusion and Future Work

Two major changes will be required to bring the brokering architecture within the OGSA framework. The first major change is the switch to using web-services

based on WSRF[23] for the implementation and SOAP[24] for the transport protocol. This will make comparatively little difference to the brokering system because that is already highly service-oriented. We expect it to simplify the communication model between the components significantly as the resource broker does not require the transfer of large amounts of data, even for usage record reconciliation. The advantage of going to a service-oriented model will be that it will become much easier to integrate UNICORE with a more traditional web-services workflow engine like BPEL[25] and so allowing more complex workflows with multiple brokering stages (e.g. a long running job that is migrated between available platforms at regular intervals, with the broker being used to select location for job migration). It will also allow a wider range of clients written in arbitrary programming languages to make use of the brokering facilities.

The second major change will be the adoption of JSDL[26] as a job description language instead of the AJO. Resource requests expressed in AJOs or JSDL documents are largely compatible, both being abstract languages that can be rendered more complete through incarnation, and both stating what resources will be required for the job being run. JSDL is a standard language for job submission (reducing the complexity of the code to map onto non-UNICORE Grids) which supports not just simple job running but also web-services invocations and database queries. This allows the broker to be used in wider settings such as load-balancing of a pool of SOAP engines or distribution of queries across a federated database.

In the future, it will be possible to create a scheduler on top of the brokering architecture outlined in this paper. This will take advantage of the fact that the VO-based broker architecture will be able to offer both a good selection of QoS offers and, through usage record monitoring, estimates of how accurate those offers are and the likelihood of those offers being honoured. In this way, the ultimate structure of the brokered scheduled Grid will probably consist of schedulers that are very close to the top (where they take advantage of the way that the brokering architecture smooths the appearance of the Grid) and the bottom (where they can take advantage of the fine information available when deciding how to match up jobs and particular resources) of the structure[3] and a brokering network in between the schedulers.

References

1. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabl;ing Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.
2. The DataGrid resource broker. `http://server11.infn.it/workload-grid/`.
3. The NorduGrid resource broker. `http://www.nordugrid.org/papers.html`.
4. Maui Scheduler. `http://mauischeduler.sourceforge.net/`.

[3] There may also be schedulers at intermediate levels if there are organizations reselling the QoS offers, acting as a clearing house for interactions. This would parallel economic activity in many other fields within and outside the computer industry.

5. NaReGI (National Research Grid Initiative) project. http://www.naregi.org/index_e.html.
6. Contract Net specification. http://www.fipa.org/specs/fipa00029/SC00029H.html.
7. WS-Agreement specification. http://www.ggf.org/Public_Comment_Docs/Documents/WS-AgreementSpecification_v2.pdf.
8. Uniform Interface to Computing Resources. http://www.unicore.org/.
9. The EUROGRID project. http://www.eurogrid.org/.
10. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, 2001.
11. The GLUE Compute Element schema. http://www.cnaf.infn.it/~sergio/datatag/glue/v11/CE/.
12. J. Brooke, D. Fellows, K. Garwood, and C. Goble. Semantic Matching of Grid Resource Descriptions. In *Proceedings of Second European Cross-Grids Conference, Cyprus 2004*. LNCS, 2004.
13. The GRid Interoperability Project. http://www.grid-interoperability.org/.
14. The Globus project. http://www.globus.org/.
15. PCPACK. http://www.epistemics.co.uk/Notes/55-0-0.htm.
16. D. Snelling, S. van den Berghe, and V. Li. Explicit Trust Delegation: Security for Dynamic Grids. *Fujitsu Scientific & Technical Journal*, 40(2), 2004.
17. The UniGrids project. http://www.unigrids.org/.
18. Condor ClassAds. http://www.cs.wisc.edu/condor/classad/.
19. Relational Grid Monitoring Architecture. http://www.r-gma.org/.
20. Network Weather Service. http://nws.cs.ucsb.edu/.
21. Usage Record Working Group. http://forge.gridforum.org/projects/ur-wg/.
22. Markets for Computational Science project. http://www.cs.man.ac.uk/cnc-bin/cnc_mcs.pl.
23. WS-Resource Framework. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
24. Simple Object Access Protocol. http://www.w3.org/TR/soap12/.
25. Business Process Execution Language. http://www-128.ibm.com/developerworks/library/ws-bpel/.
26. Job Submission Description Language. http://forge.gridforum.org/projects/jsdl-wg/.