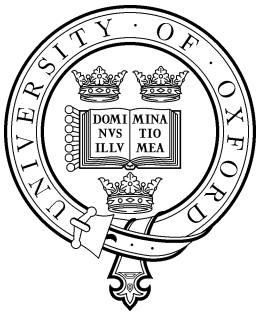
Programming Research Group

OBJECT MODELS: JOB SUBMISSION IN DATAGRIDS

Lee Momtahan and Andrew Martin

PRG-RR-04-26



Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Abstract

DataGrids seek to pool geographically disperse storage and computational resources to solve large-scale data-intensive problems. The EU-DataGrid is a large research project attempting to build one of the first such DataGrids. Effective job submission and scheduling in DataGrids is a a significant challenge. We present an account of the EU-DataGrid scheduling architecture, some typical usage scenarios, and a discussion of some of the issues that arose when this architecture was used in practice. Some of these issues are generic, and likely to be of interest to future attempts to build a DataGrid, but also of a pragmatic nature, illuminated by experience as well as theory.

1 Introduction

The EU-DataGrid (EDG) is one of the largest attempts yet to build a DataGrid [Seg00] with 200 researchers involved from over 20 institutes across Europe for a period three years. The project is designed to provide the computational infrastructure required for the next generation of science in the areas of high energy physics, bio-informatics and earth observation. Grids are characterized by large-scale heterogeneous storage and compute (cluster and supercomputing) resources, massive geographical distribution, and potentially large user communities organised into overlapping dynamic virtual organisations.

The scheduling of user compute jobs upon these various resources — with suitable access to storage, data sets and so on — is a significant challenge: the design of the scheduling system is central to the realization of the distributed resource as a virtual massive computing device. Conversely, a poor job submission scheme has the potential to render the rest of the architecture worthless.

In this paper, we explore the dimensions of the problem. We briefly consider what makes scheduling in a DataGrid context a different problem from the relatively well-understood problem of scheduling in supercomputing and cluster computing. In Section 3 we explain the architecture adopted by the EDG, and in the following section present the principal usage scenarios supported by the software, illustrated by sequence diagrams. Section 5 broadens the discussion to consider in detail the more generic issues which have been encountered (whether or not supported by EDG) in consideration of those use cases. The paper ends with some conclusions on how those complexities may be incorporated into an EDG-like Grid.

In reflecting upon the modes of use for the EDG we have found it instructive to use UML models to describe the static and interacting structure of the distributed components. Our purpose in writing is to report upon the design of the EDG documented in that way, and to provide a concise starting point for anyone wishing to understand, deploy, or adapt the EDG architecture.

2 Job submission and scheduling in DataGrids

Job scheduling in distributed computing is a well-studied topic and has been thoroughly described [CK88]. Scheduling in Grids presents new challenges since the scheduler does

not have full control and ownership over the resources in a Grid. Moreover, there may not be a single scheduler (this becoming a distributed function of the collective layer), and the schedulers may have to rely on partial or out-of-date information. In *DataGrids* even more challenges arise since inter-dependent compute and data tasks must be scheduled. It is often the case that requirements for data files can only be known during the execution of a job.

Berman notes that Grids violate the assumptions in traditional massively parallel processor schedulers with resource pools which are heterogeneous, dynamic, shared and under multiple administrative domains. Furthermore, Grid schedulers have to cooperate with other Grid level and lower level schedulers and schedule different resources simultaneously [Ber99].

Schopf [Sch01] defines super-scheduling as scheduling across multiple administrative domains, and usefully outlines the steps executed by the EDG's resource broker However the interaction of competing super-schedulers and the optimal co-allocation of compute and storage resources is not considered. The Global Grid Forum's Grid Resource Allocation Agreement Protocol Working Group is defining the standards and protocols to support negotiations between a Super-Scheduler and local scheduling systems.

In [TK04] Kosar and Livny propose Stork, a scheduler for data transfers in the Grid. Stork interacts with DAGMan, a higher level planner, allowing the user to define dependencies between data transfers and computational tasks in their application. This allows the automatic staging in and out of data. Although a massive improvement over manual data transfer, Stork works independently from other schedulers (except for the necessary temporal coupling imposed by DAGMan); data locality is not taken into account during job placement. Kosar and Livny leave co-scheduling of computational and data resources as a topic of future research.

Some of the issues relating to replica selection are addressed by Vazhkudai et al [VTF01]. They describe a selection algorithm for applications to use at run time, ranking individual replicas. Ranganathan and Foster [RF02] provide a simulation (in the context of the GriPhyN project) of various replication strategies. Later [RF03], they use further simulations to show that job scheduling must take into account data placement and conclude that good performance can be achieved by scheduling jobs where data is located, with a separate process periodically creating replicas at sites where they are most in demand.

3 The Architecture of the EDG

Here we describe features of the EDG which are relevant to our present discussion. The term 'EDG' is slightly ambiguous in that properly the term refers to a suite of software developed using version 2 of the Globus toolkit [FK99, chapter 11], and also Condor [TTL02], but the main experience of the EDG to date has been its running *Testbed* which has been aiming to offer something close to a 'production Grid' service through the latter stages of the project. In future, the EDG team expect to see the software deployed in a number of distinct situations.

The EDG high level design presented here was captured through a process suggested by one of the authors providing methodology advice to the EDG's Architecture Task Force. Tracking the overall design has been difficult for the ATF because software development proceeds across a large number of institutions distributed over Europe [MM02]. The ATF contains representatives from each of the main software development teams. In the design exercise, each representative chooses an abstraction of their software into one or more components. Application scientist representatives choose a test case or scenario, and the resultant sequence(s) of interactions between components which would occur in the execution of the test is discussed by the group. This is very much in the style of the well-known Class-Responsibilities-Collaborators exercise [BC89]. A more complete exercise was not possible due to the limited time available for ATF meetings.

For the job submission scenarios we have further abstracted away from concerns not relevant to the present discussion and presented the pattern of interaction in the spirit of the UML [FS97] in section 4. Such diagrams helpfully show the various components taking part in the process, *but do not necessarily cover all the exceptional or unexpected behaviours*. The diagrams are supported by prose explanations of each interaction. Firstly though, we describe the components themselves.

3.1 Components

We introduce some of the terms used to describe elements of the DataGrid. EDG has common terms which we define here. Other projects may differ in their terminology, but we expect similar concepts.

Storage Element This is the name given to a storage resource available to the Grid. The Storage Element supports functionality to store and retrieve files. Also, the Storage Element can give an estimate of the time taken to retrieve a file. The Storage Element publishes its status (for example the amount of free space) into the Information System.

Computing Element This is the name given to a computational resource available to the Grid. Typically Computing Elements present to the Grid an interface for a Local Batch System. Jobs can be submitted to the Computing Element and their status queried. The Computing Element publishes its status (for example the number of free CPUs, queue length, etc.) into the Information System.

User Interface This component runs on the end-users' local machines and provides them with an entry point into the Grid, from which they can submit jobs. It can also be installed within Computing Elements to allow jobs to be submitted by other jobs (see 4.4), not just users.

Information System Computing Elements and Storage Elements periodically update the Information System with their presence and status. The Information System is queried by Resource Brokers to help make scheduling decisions. Because the Information System is updated only periodically, and caches and aggregates data sent to it, the information returned may not be up to date.

Resource Broker The Resource Brokers are responsible for finding the best Computing Element on which to run a particular job. To help the Resource Broker make this decision a Job Description is provided when submitting a job which contains information about the resources needed to execute the job and other hints.

Physical File A Physical File refers to some particular instance of a piece of data on the Grid. Physical Files are stored in a Storage Element. Physical Files may be *replicated* — identical copies made, and stored at different locations — hence the need for *Logical Files*.

Logical File A Logical File refers to a piece of data on the Grid. Each Logical File refers to one or more *Physical Files*, all of which have identical content.

Replica Manager A Replica Manager maintains the mapping from Logical to Physical File. For every Computing/Storage Element and Physical File it can associate a cost (i.e. time) to transfer the file to the Computing/Storage Element. Given a Computing/Storage Element and Logical File it can find the least costly Physical file for the Computing/Storage to access.

Job Description The Job Description is used to inform the Resource Broker of a job's requirements (for example, operating system type, memory required, and so on). Is also includes a list of the names of the Logical Files needed by the Job. Not all jobs will be able to use remote access protocols, and because this can affect job submission there is a flag for this.

Figure 1 is a class diagram showing the static relationships between the entities we have described above. In this diagram, the association between Logical File and Physical File, has the 'Replica Manager' stereotype to indicate that traversing this link requires a call to the Replica Manager.

3.2 DataGrid Testbed

The issues we report later in this paper were seen in practice whilst the EDG software was running on a testbed, a quasi-production Grid of the following approximate scale:

- about ten sites (mostly European but some others) in the collaboration;
- hundreds of CPUs (worker nodes) and terabytes of disk storage;
- several resource brokers;
- scores of users; and
- several Virtual Organisations.

More information on the testbed setup can be found in the Testbed Evaluation [ED03].

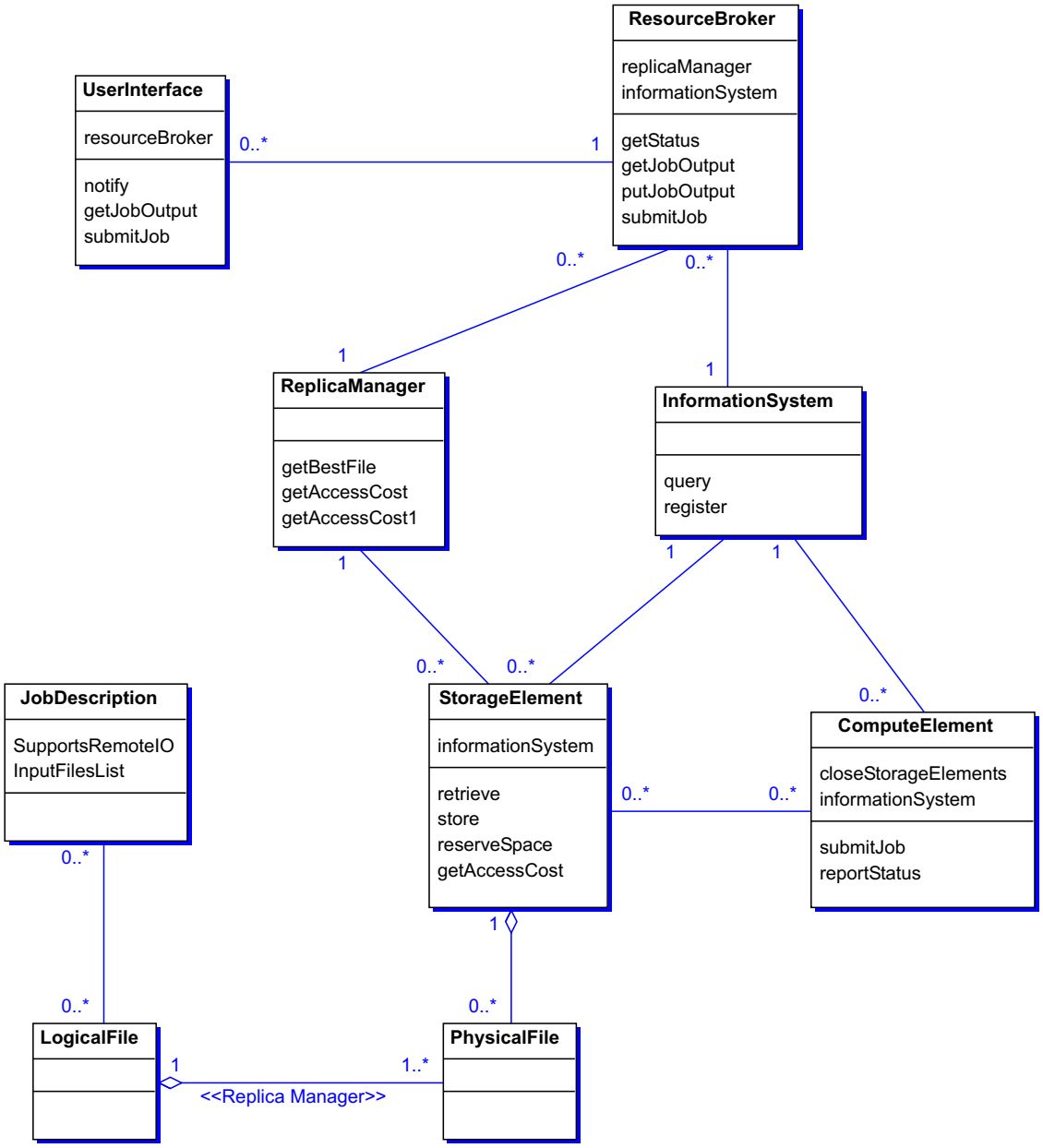


Figure 1: Class Diagram

4 Job Submission scenarios in the EDG

We explore here the main job submission scenarios supported by the EDG software. There are three special cases which illustrate the features of the available systems [ED02]:

- job submission with no data requirements—no input files e.g. Monte Carlo simulation;
- job submission with static file selection—where the input files can be determined ahead of time; and
- job submission with dynamic file selection—where the file selection must happen at runtime.

We also describe the fetching of job *output* data. As explained above, the resource broker is responsible for receiving user job submissions and selecting compute resources on which they are to run. When a job completes, the compute element returns to a cache on the resource broker the output which it generated. The end user must retrieve this output data from the resource broker, so we also explain how this is achieved.

There is also a process whereby Computing Elements, Storage Elements and Network Monitors periodically update the Information System with their their presence. This process is often called 'soft-state registration' since after a configurable timeout period, if the Information System has not received an update from a particular resource it assumes the resource is no longer available. This process is an ongoing or *background* process; it is not triggered by a user action. We have not presented a diagram for this process but mention it here for completeness.

4.1 Job Submission with no data requirements

The simplest imaginable job is one which does not need any input files. This may appear to be an over-simplification, but is commonly the case for Monte Carlo simulation jobs in High Energy Physics for example. Very small amounts of input data — the values of parameters in a simulation or a random seed — can be expressed in the job description and easily passed to the job executable. The simplicity of this pattern (and its repetition in the other interaction cases) makes it worth recording. The output data (or a pointer to it) is stored on the resource broker; the next case covers its recovery by the user.

1 The User creates a Job Description and issues a *submit-job* command on their User Interface.

1.1 The User Interface forwards this request on to the Resource Broker.

1.1.1 The Resource Broker queries the Information System to find the status of all available Computing Elements.

1.1.1.1 The Information System asks each Computing Element to report its status unless it already holds of a recent version in cache.

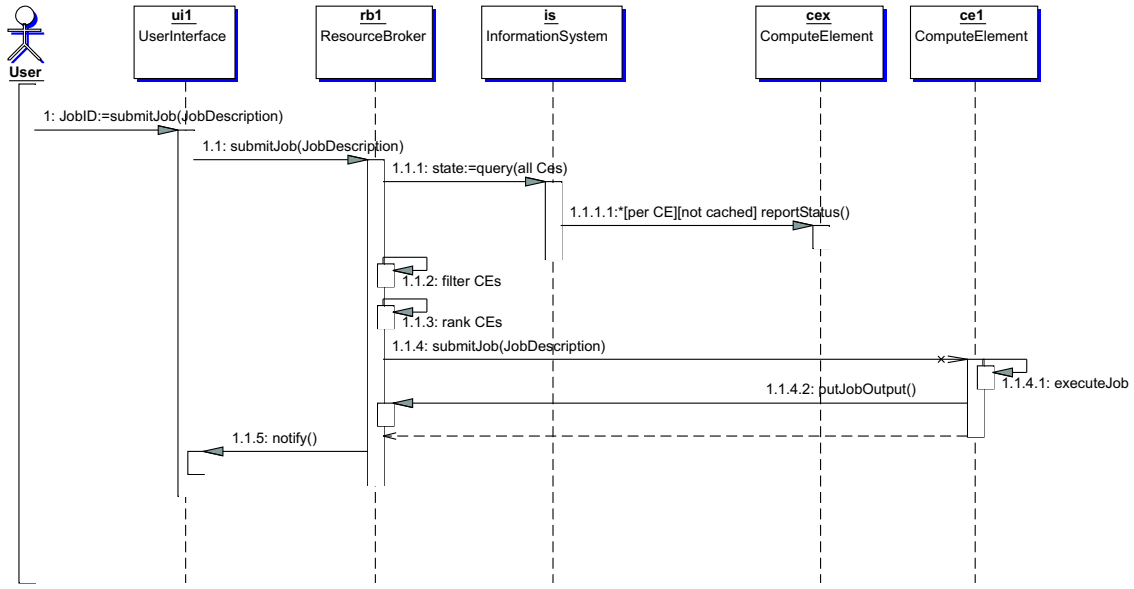


Figure 2: Simple Job Submission

1.1.2 The Resource Broker obtains a list of candidate Computing Elements by filtering only those which match the requirements specified in the Job Description.

1.1.3 The Resource Broker ranks the Computing Elements. A default formula is used for this, but can be overridden in the Job Description.

1.1.4 The Resource Broker sends job to be executed on the top ranked Computing Element.

1.1.4.1 The Computing Element runs the job.

1.1.4.2 The output of the Job is stored on the Resource Broker.

1.1.5 On completion of the job the Resource Broker notifies the User, via email.

4.2 Fetch Job Output

Job output data is stored at the Resource Broker (it cannot be stored at the Computing Element which ran the previous job, because these are unknown to the user). Normally this will be the *standard output* of the job and a relatively small amount of data. For jobs which will produce vast amounts of output data, it is recommended that the job saves the bulk output on a Storage Element, especially if this output will require further processing on the Grid; the mechanism described here is then used to return a pointer to the bulk output.

This use case describes how users may fetch the output data produced by a previous job.

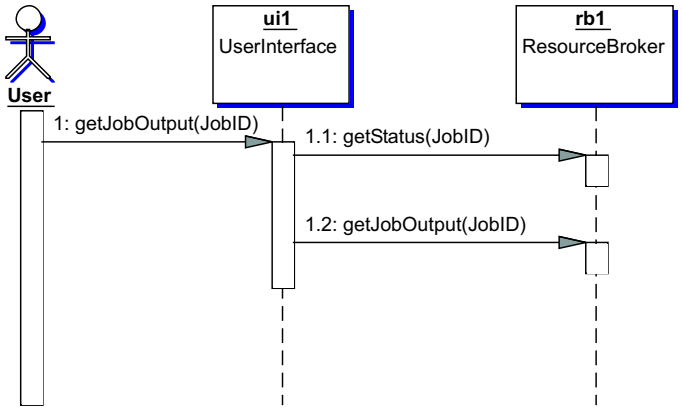


Figure 3: Fetch Output

1 The User enters a command on their User Interface asking to fetch the job output. The Job Identity is specified to determine which job.

1.1 The User Interface checks with the Resource Broker that the job has completed and its output has been received.

1.2 If so, the output is moved to the User Interface.

4.3 Job Submission with Static File Selection

A more complicated scenario is the submission of a job which requires input files. In this subsection we assume that the input files can be determined at submission time; the next deals with the case when the files can only be determined at run time.

The task of the Resource Broker is now more complex in that it must collect information about available replicas of the logical files in question, and possibly reserve space at a local storage element for a further copy.

1–1.1.2 The steps are the same as in the case: *Job Submission with no data requirements* - subsection 4.1.

1.1.3 The Resource Broker calls the Replica Manager with its list of candidate Computing Elements and the Logical files needed by the job.

1.1.3.1 Having determined all the physical instances of the Logical Files, the Replica Manager asks each relevant Storage Element to estimate the cost of retrieving the files.

1.1.3.2 The Information System is queried to estimate the bandwidth of the relevant networks so that the file transfer time can be added. Thus the Resource Broker is returned a matrix of Computing Elements and the costs with each Logical file.

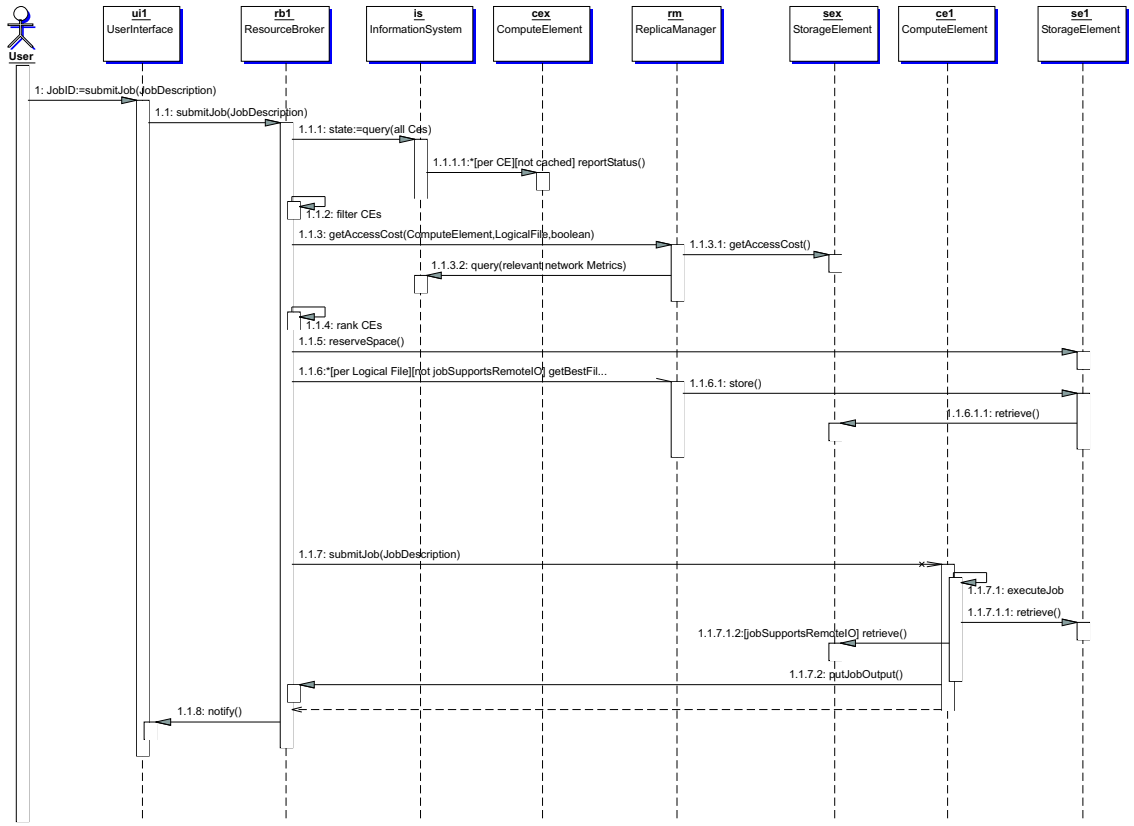


Figure 4: Job Submission with Static File Selection

1.1.4 This matrix, along with the state of each Computing Element is combined to derive an overall ranking for each Computing Element. The exact formula is tunable and the default can be overridden.

1.1.5 Since files may need to be copied to a Storage Element local (i.e accessible via POSIX) to the Computing Element this call checks for necessary space. (When advance reservation is supported — see 5.3 — the call will reserve the necessary space.) If the space is not available, the next highest ranking Computing Element is chosen and this step is repeated.

1.1.6–1.1.6.1.1 The Resource Broker asks the Replica Manager to copy each Logical File to the local storage element (*se1* in the diagram) in the case that the job does not support remote IO.

1.1.7 Now the Resource Broker can submit the job to the Computing Element

1.1.7.1–1.1.7.1.3 The job runs retrieving any necessary input files.

1.1.7.2 The job's output is stored on the Resource Broker.

1.1.8 On job completion the Resource Broker notifies the User via email.

The relatively complex interaction pattern, and the long-lived nature of the copying process, give rise to numerous possible failure modes. See Section 5.7.

4.4 Job Submission with Dynamic File Selection

In this scenario the job needs some input files, but exactly which files cannot be determined until run-time. For example it may be that the files needed for input can only be determined after some initial processing. For simplicity we have assumed here that no Logical Files are specified in the Job Description, but in fact this case can be combined with the previous one in the obvious way: some Logical Files can be specified at job submission, and some decided at run time.

1–1.1.4 The job is submitted to the Computing Element using exactly the same steps as in the case: *Job Submission with no data requirements* - subsection 4.1.

1.1.4.1 The job is started on the Computing Element.

1.1.4.1.1 The job decides it needs a Logical File and asks the Replica Manager to determine the best replica available.

1.1.4.1.1.1 The Replica Manager determines all the physical instances of the Logical File and asks each relevant Storage Element how long it will take to retrieve the physical file.

1.1.4.1.1.2 The bandwidth between the Computing Element on which the job runs and these Storage Elements is obtained from the Information System because it is used to determine the best physical file.

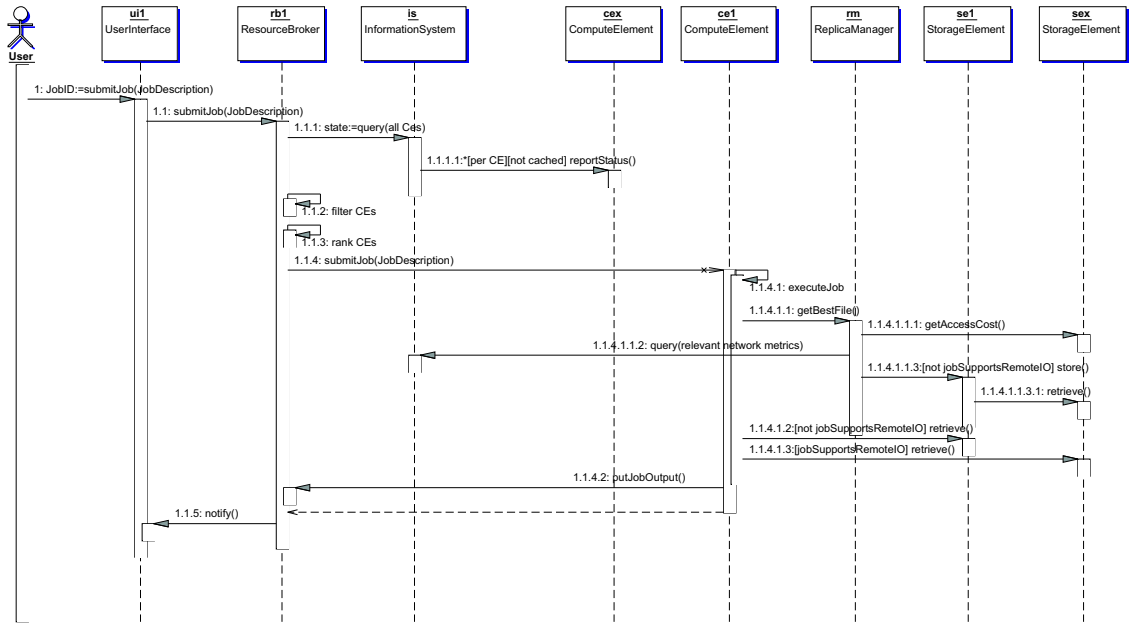


Figure 5: Job Submission with Dynamic File Selection

1.1.4.1.1.3 The Replica Manager also triggers a replication of this best file to the local Storage Element in the case that the job does not support remote IO.

1.1.4.1.2 The job retrieves the file from the local Storage Element if it does not support remote IO.

1.1.4.1.3 The job retrieves the file from the remote Storage Element otherwise.

1.1.4.2 The output of the Job is stored on the Resource Broker.

1.1.5 On completion of the job, the User is notified via email.

At 1.1.4.1.1.3 or 1.1.4.1.2 we note that the job may block on the file transfer and this might lead to inefficient use of the Local Batch System resources. To avoid this the job may instead of executing 1.1.4.1.1, save its state in a Storage Element, and submit a fresh job to the Grid. This new job can have the required files in its Job Description, and we have a case of Job Submission with Static File selection. With this new information the Resource Broker may decide to send the job to a Computing Element closer to the required data. In effect this allows moving the job to the data rather than moving the data to the job. Of course the upside of more efficient input data access needs to be balanced with the downside of having to save and reload state plus resubmitting the job and letting it traverse the queue of the Local Batch System on which it is scheduled.

One might argue that this alternative of job resubmission using static file selection is always favourable because it avoids wastage of compute resources. The situation seems akin to the potential wastage of compute resources in non-Grid local batch systems where

jobs may need to wait for files to be spooled from tape. We would expect that policies which have been evolved for those situations to be applicable in the Grid context also. The correct trade-off is an application-specific matter, depending how much computation is performed before the job needs new input files.

5 Issues

In the preceding section we have presented the principal job submission scenarios supported by the EDG. In describing these, many issues have been considered, and we document them here. Addressing most of these would introduce too much complexity to the present code base. In some cases, they are sufficiently poorly understood that it would even be hard to simulate their effects. Nevertheless, they indicate the complexities which must be managed if wide-use production grids are to become a reality.

Not least among the issues is the the need to support economic models of Grid use. Some patterns of utilisation simply will not come about until there is a manageable 'cost' of using the resource; others can only sensibly be put in place if there is the feedback of a charging mechanism as a control: for example, the implementation of *advance reservation* is an invitation to reserve far more resources than are likely to be needed, for far longer than necessary—whereas efficient Grid utilisation will arise only when this is constructed as a form of futures contract.

5.1 Legacy Fabric Components

The Mass Storage Systems and Local Batch Systems which make up the DataGrid 'production testbed' are valuable resources which are used in production today. It is not realistic to expect Data Centres to uproot a software infrastructure running their critical applications. Dedicating hardware to a particular Grid is expensive, may lead to resource under-utilisation in the short term, and does not form an appealing migration strategy.

However, many aspects of these components are not ideal in their new role as Grid fabric components. For example, most of the worker nodes in today's farms do not have outbound access to the Internet for reasons of security or network addressing. Only when these nodes are used in a Grid context, where running jobs may need to go outside the farm to access remote Grid services, is this downside realised. It is yet to become clear whether the Grids will have to overcome this difficulty, or whether the policies on these farms will change.

5.2 Data Prefetching

Another complication is the use of local batch systems with data stored remotely. The execution of a job at a site may require a large amount of data to be staged there, and it is unclear how this should proceed. Let us consider the options:

If we wait for the job to traverse the queue of the Local Batch System before triggering the replication of data there is the possibility that the job will make inefficient use of the Local Batch System resources whilst waiting for the data to arrive.

Another strategy is to trigger the transfer of data exactly when the job is submitted to the Local Batch System. But again, the job may traverse the queue before the data transfer completes. Either the job ties up the resources of the Local Batch System while waiting again, or it resubmits itself to the back of the queue, impacting the response time of the job. In the latter case the arrival of new jobs on this Local Batch System, since the job was first submitted, may impact response time even further. If the job takes much longer to traverse the queue than it takes to transfer the data, we end up with local storage resources being used inefficiently, or even filling up completely.

A further alternative would be to trigger staging and wait for this to complete before submitting the job to the site e.g.[TK04]. Again there is a danger local storage resources could be used poorly.

In isolation, this issue may be solved by use of more elaborate multi-level queues, but in combination with the more general problems of advance reservation (below) and scheduling at both the Grid and the local batch level, the general case of the problem becomes very hard to solve.

Unfortunately EDG has did not develop a POSIX interface to for remote data access and many applications therefore require data to be staged. This may not always be suitable, but would alleviate many problems.

5.3 Advance Reservation

Currently the Storage Elements on the EDG do not support advance reservation. Even though a Job Description may state that a job requires a certain amount of local storage this is checked only at the time of job submission; there is the possibility that this capacity will be unavailable by the time of job execution.

Whereas advance reservation is not critical for cpu and network resources — without it jobs will still receive some slice of the available capacity — it seems that it can be essential for storage resources (for staged input, intermediate results, or final output). Nevertheless without reservation in these other resources it might be difficult to know how for long a storage reservation should be made.

A spectrum of advance reservation possibilities can be described:

- There is the present EDG implementation, with no advance reservation on any type of resource. Free storage is checked at job submission time; the job is retried if the storage is used up before the job completes. Check-pointing features prevent the job being re-executed entirely.

- Alternatively, there could be advance reservation on storage resources only, making an advance reservation at job submission time for the expected duration of the job, attempting to renew the reservation if the job takes too long, but falling back on retrying the job again.

- At the other end of the spectrum we could permit advance reservation on all types of resources, making advance reservations on all relevant resources, minimizing the chance of overrun and the need for re-execution.

5.4 Optimal Scheduling

In a production Grid context, the heterogeneous nature of users, jobs, and resources complicates any notion of what it might mean to be optimal. For most users, a good scheduler will be one which delivers short turn-around times for jobs; for resource owners, it will be one which leads to high utilization. The best outcome for particular users may be poor for the system as a whole. In a Grid context the trade-off is further complicated by the presence of the virtual organizations — the throughput for the VO relative to the whole resource may be of interest, as may the response for an individual relative to the VO.

Therefore, it is not clear what kind of *scheduling policy* is required. Maybe all Virtual Organisations (VOs) and their members will work together to optimise the overall throughput of the Grid. Or maybe each user (or VO) works independently to minimise the response time of their own jobs.

Other factors influence the scheduling policy:

One or many resource brokers The number of resource brokers present in the system is critical. With exactly one Resource Broker there is the opportunity to make globally optimal decisions. But, ironically, a single Resource Broker will itself become a bottleneck as the Grid scales.

Multiple Resource Brokers will be closer to optimal, even if they do have to work independently so that the communication between them does not become a bottleneck. In this case we need to be wary that information coming from the Information System is likely to be out of date - the action of one Resource Broker on the Grid will not be seen by other Resource Brokers until some time later due to caching in the Information System - and in practice we found this leads to a *thrashing* effect as described in [Ber99]. That is, when multiple independent Resource Brokers submit jobs into the Grid at a sufficient rate, resources which appear to be unloaded according to the Information System may received too many jobs. This experience has lead us to believe that scheduling with a degree of randomness will behave more reliably. This observation is further supported by the theoretical results in [RF03], and the experiences in [Ell02].

Information availability Another question is how much data should be published in the information system. The current model used in the EDG is for the Computing Element to publish all the information the Resource Broker might need from it into the Information System. This model has the advantage that the Resource Broker can find the best Computing Element for a job just by contacting the Information System. However if a Computing Element has a different scheduling policy per VO, the amount of information to be published may become prohibitive.

An alternative model might say that only a small amount of fairly static data should be published, and once the Resource Broker has made a first pass to filter unsuitable Computing Elements, it must contact them directly. Although the Resource Broker has to directly query many Computing Elements now, each Computing Element

could have its own algorithm to estimate how quickly it can execute the job. This also has the advantage that for highly dynamic data, the Resource Broker works with fresher information.

Highly distributed scheduling An entirely different model has been proposed [EKS+03] in which each user effectively has their own resource broker. Effectively submission is random but weighted toward the highest ranked nodes. This seems to work and scale.

5.5 Job Description

The form of the job descriptions will be important as a they are key to most scheduling decisions. Constraints on the type of computational resource (chip architecture, operating system type, and so on) are commonplace and can be used immediately to filter out non-matching resources. One might also specify which files are required by the job, and this information can be used to provide hints to the scheduler. But if only a small fraction of file is used the hint may be misleading. Whether a job processes its specified input files in serial or parallel, and whether the files are read serially or randomly are factors to which any scheduling decision may be very sensitive. Not only do we need to consider what hints are useful to scheduling, but what hints the end-user has the ability to provide.

5.6 Base metric

Some have suggested elaborate techniques for performance optimisation in a DataGrid based on economic models, but it is not clear yet whether the base metrics assumed by there models can be realised. For example, which of two mass storage systems/tape robots is preferable to serve a large dataset may come down to which can recover the data from tape the quickest at that point. In practice making such estimates is exceedingly hard, and may not be viable. The appropriate economic models will depend on a number of factors [Mom].

5.7 Fault Tolerance and Exception Handling

In this paper we have only been able to look at the basic flows relating to job submission in EDG; there are a large number of devious flows which should also be supported.

This is particularly the case for fault tolerant behaviours. Although conventionally treated as exceptional, fault tolerant behaviour becomes 'business-as-usual' in Grids — as the number of nodes in the deployment scales up there is almost always a faulty node or network link somewhere.

Such flows may necessitate a different design: the appropriateness of the present design can really only be judged by observation of how and why failures occur. Such data is not presently available in the testbed status report [ED03].

6 Conclusions

The experience of the EDG has shown that there are many dimensions to the problem of job submission and scheduling, and only a few have been considered in the present code base. Experience with the EDG testbed casts doubt upon the use of a small number of resource brokers: the scalability of such a solution seems unlikely. Instead, schemes with more resource brokers — or a radically different approach — are under consideration.

We have documented the main features of the present software, with due care given to the possible failure modes. We have also shown above that many more factors might be taken account of in the design of schemes for job submission in data Grids. Several of these are non-trivial: they have potential to impact severely on the throughput, utilisation, and turn-around time when a large-scale heterogeneous production Grid is implemented.

Those considerations will be critically dependent on patterns of use. It is inappropriate to try to solve the general problem at this stage because there are too many unknown parameters. We expect that the forthcoming deployments of DataGrid software will provide an excellent opportunity to measure realistic patterns of usage and to identify sources of inefficiency. The discussion in this paper may help to target that effort.

References

[BC89] Kent Beck and Ward Cunningham. A laboratory for teaching object-oriented thinking. In *Proceedings of OOPSLA 1989*, 1989.

[Ber99] Francine Berman. High-performance schedulers. pages 279–309, 1999.

[CK88] T.L. Casavant and J.G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, February 1988.

[ED02] EU-DataGrid. The DataGrid architecture. Technical Report DataGrid-12-D12.4-33671-3-0, EDG, 2002.

[ED03] EU-DataGrid. Final evaluation of testbed operation. Technical Report DataGrid-06-D6.8-414712-3-0, EDG, 2003.

[EKS+03] P. Eerola, B. Kónya, O. Smirnova, T. Ekelöf, M. Ellert, J. R. Hansen, J. L. Nielsen, A.Wäänänen, A. Konstantinov, and F. Ould-Saada. The NorduGrid architecture and tools. In *Computing in High Energy and Nuclear Physics 2003, La Jolla, California*, March 2003.

[Ell02] Mattias Ellert. The nordugrid toolkit user interface and resource broker. Presented at the Fourth NorduGrid workshop, Uppsala, November 2002. http://www3.tsl.uu.se/ ellert/publ/NorduGrid-Uppsala.pdf.

[FK99] Ian Foster and Carl Kesselman, editors. *The Grid.* Morgan Kaufman, 1999.

[FS97] Martin Fowler and Kendall Scott. *UML Distilled: Applying the Standard Object Modelling Langauge*. Addison-Wesley, 1997.

[MM02] Lee Momtahan and Andrew Martin. e-science experiences: Software engineering practice and the EU DataGrid. In *Asia-Pacific Software Engineering Conference, IEEE Press (APSEC 2002)*, pages 269–275, 2002.

[Mom] Lee Momtahan. Grid economics: Standardised grid resource commodities. (Work in progress).

[RF02] Kavitha Ranganathan and Ian T. Foster. Identifying dynamic replication strategies for a high-performance data grid. In *International Workshop on Grid Computing*, 2002.

[RF03] Kavitha Ranganathan and Ian T. Foster. Simulation studies of computation and data scheduling algorithms for data grids. *Journal of Grid Computing*, 1:53–62, 2003.

[Sch01] J. M. Schopf. 10 actions when superscheduling, July 2001. Global Grid Forum Document GFD-I.4.

[Seg00] Ben Segal. Grid computing: The european data grid project. In *IEEE Nuclear Science Symposium and Medical Imaging Conference, Lyon, France*, October 2000.

[TK04] Miron Livny Tevfik Kosar. Stork: Makring data placement a first class citizen in the grid. In *Proceedings of 24th IEEE Int. Conference on Distributed Computing Systems (ICDCS2004), Tokyo, Japan*, March 2004.

[TTL02] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.

[VTF01] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the globus data grid. In *International Workshop on Data Models and Databases on Clusters and the Grid (DataGrid 2001)*. IEEE Computer Society Press, 2001.

Acknowledgements

The EDG is a large project and many have contributed to the design and code of the present system. We are grateful to the members of the Architecture Task Force (especially: Franck Bonnassieux, Akos Frohner, Leanna Guy, Jens Jensen, Erwin Laure, Julian Linford, Cal Loomis, German Melia, Johan Motagnat, Fabrizio Pacini, Piotr Pozanski, Jeff Templon, Annalisa Terracina, Antony Wilson) for discussions which led to our present understanding, and for their participation in the construction of the sequence diagrams presented here. We would also like to thank Ian Stokes-Rees for his comments and proof reading.

This work has been supported by the UK research councils PPARC and EPSRC under EPSRC grants GR/R74284/01 and GR/S73204/01.